INFO-F101 : Programmation
Syllabus d'exercices

Réalisé par l'équipe d'assistants du cours INFO-F101

Département d'Informatique, Faculté des Sciences
Université libre de Bruxelles

Édition 2013–2014
« Chuck Norris »

Remerciements et historique

Édition 2013–2014 (réorganisation des exercices, création de nouveaux exercices). Liran Lerman, Luciano Porretta et Nikita Veshchikov (coordinateur du chantier), Mohamed Amine Youssef.

Édition 2012–2013 (réorganisation des exercices, création d'exercices préparatoires, passage à Python 3). Liran Lerman, Markus Lindström (coordinateur du chantier), Luciano Porretta et Nikita Veshchikov. Merci également à Marie Boulvain (conseillère pédagogique du Centre de Didactique Supérieure de l'Académie universitaire Wallonie-Bruxelles) pour l'aide à la rédaction de la préface.

Édition 2011–2012 (création chapitre « Instructions de base » et compléments sur complexité et invariants). Stéphane Fernandes Medeiros, Markus Lindström, Naïm Qachri et Alessia Violin.

Édition 2010–2011 (« Big Bang » : passage de C++ à Python 2). Stéphane Fernandes Medeiros, Naïm Qachri et Jérôme Vervier. Relecture par Markus Lindström et Alessia Violin. Remerciements également à Hadrien Mélot (UMons) pour nous avoir laissé nous inspirer des exercices de son cours pour réaliser cette édition.

Les exercices du chapitre « Instructions de base » introduit dans l'édition 2011–2012 ont été inspirés par le syllabus du cours de Programmation (anciennement « Algorithmique et Programmation ») des éditions précédentes, orientées vers le langage C++, auquel ont contribué Nadjet Benseba, Emmanuel Dall'Olio, Martin De Wulf, Gilles Geeraerts, Joël Goossens, Christophe Macq, Olivier Markowitch, Thierry Massart et Patrick Meyer. Cet ouvrage a également servi de base principale pour les chapitres « Complexité » et « Logique et invariants ».

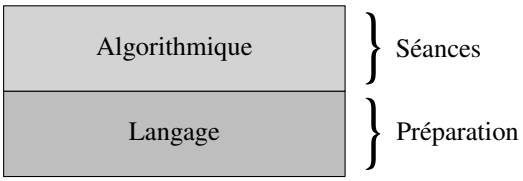
Informations importantes

Les séances d'exercices de programmation servent plusieurs objectifs pédagogiques :
– Maîtriser les bases du langage de programmation Python dans des mises en situation ;
– Développer l'esprit logique et la capacité à réaliser des algorithmes ;
– Mettre en pratique quelques bases d'informatique théorique (complexité, logique et invariants) ;
– Inciter l'étudiant à travailler régulièrement.

Chaque chapitre du présent syllabus est séparé en deux parties :

Préparation. Chaque chapitre exige l'étude préalable de *prérequis* (typiquement, de la matière vue au cours, ou éventuellement vue en secondaire) ainsi que la résolution éventuelle *d'exercices préparatoires*. Ceux-ci forment une base pour la réalisation des exercices en séance. L'objectif de la préparation est de faire acquérir à l'étudiant les éléments nécessaires de *syntaxe* et de *sémantique* du langage de programmation utilisé.

Séance(s). Les exercices en séance sont encadrés par des assistants. Ces derniers constateront si l'étudiant a correctement préparé chaque séance et s'ils peuvent dès lors proposer des problèmes *d'algorithmique* (c'est-à-dire, étant donné les outils fournis par le langage de programmation et leur mise en œuvre, comment peut-on résoudre efficacement un problème donné ?).



Les motivations sous-jacentes à cette découpe sont multiples :
– L'expérience nous a montré que si les étudiants ne maîtrisent pas la syntaxe et la sémantique du langage utilisé, ils ne sont d'une part pas en mesure d'écrire des algorithmes dans ce langage, mais d'autre part ne sont pas non plus armés pour comprendre les explications fournies par les enseignants vu l'absence de vocabulaire commun.
– Les rappels théoriques autrefois donnés en début de TP ne semblaient pas servir d'objectif concret : les étudiants ayant déjà compris la matière perdaient leur temps, tandis que les autres avaient souvent trop de lacunes que pour pouvoir utiliser efficacement l'information fournie. La préparation de la matière avant chaque séance d'exercices devrait pallier ce problème, tout en permettant aux assistants de réduire le temps passé pour des rappels théoriques et d'ainsi pouvoir se concentrer sur la résolution d'exercices et l'acquisition de compétences en algorithmique.

– La préparation individuelle des chapitres confronte l'étudiant à l'interpréteur Python et lui permet de tester son comportement réel, condition *sine qua non* pour la réalisation d'exercices et la réussite du cours.

Les séances sont construites autant que possible pour que les exercices aient une difficulté croissante. Les exercices dénotés par une astérisque * sont réputés de niveau examen, tandis que ceux dénotés par deux astérisques ** sont de difficulté supérieure à ce qui serait demandé à un examen. Ces derniers exercices ont pour but de permettre aux étudiants le souhaitant de se dépasser.

Règles du jeu

L'étudiant est tenu de préparer chaque chapitre de ce syllabus de manière individuelle en s'aidant du cours théorique. Les exercices préparatoires forment un prérequis nécessaire à la compréhension et à la participation aux séances d'exercices. Si l'étudiant rencontre des difficultés qu'il estime insurmontables dans le cadre des exercices préparatoires, il est invité à contacter l'assistant de son groupe par courrier électronique au moins 24 heures avant la séance (compter un jour ouvrable).
Il est demandé à l'étudiant de respecter le travail de ses condisciples aux séances.
Les corrections d'exercices ne seront pas systématiquement fournies, cela afin d'éviter que l'étudiant vienne aux séances uniquement pour recopier les solutions. L'étudiant aura à sa disposition une plateforme interactive d'apprentissage, nommée *UpyLaB*, lui permettant de tester pour chaque exercice fourni son propre programme et d'avoir une indication sur la validité de ce dernier. La plateforme est accessible à l'adresse suivante :

`http://upylab.ulb.ac.be/`

Coordonnées des assistants

L'étudiant peut toujours contacter l'assistant de son groupe en dehors des séances d'exercices en fixant un rendez-vous préalable par courriel. Les coordonnées des assistants sont données par chacun d'eux lors de la première séance et sont également disponibles sur l'Université Virtuelle :

`http://uv.ulb.ac.be/`

Des versions électroniques du présent syllabus d'exercices ainsi que des supports de cours et d'anciens examens sont disponibles à cette même adresse.

Guidances

Des guidances sont à disposition de l'étudiant à partir du mois d'octobre. Toutes les informations relatives à ce service sont disponibles à l'adresse suivante :

`http://www.ulb.ac.be/di/guidances/`

Table des matières

1 Séance introductive **1**

2 Contrôle de flux **4**

3 Fonctions **11**

4 Chaînes de caractères **17**

5 Listes **22**

6 Matrices **26**

7 Dictionnaires **30**

8 Tris **35**

9 Complexité **37**

10 Logique et invariants **42**

11 Récursivité **48**

12 Fichiers et exceptions **50**

13 Introduction aux classes **54**

14 Base de données **57**

15 Révision **61**

A Swampy Turtle **66**

B Python Imaging Library **69**

C Aide-mémoire Python **72**

Chapitre 1

Séance introductive

Exercices préparatoires

Matière à réviser :
– les variables ;
– les fonctions `print()` et `type()` ;
– les opérateurs `+`, `-`, `*`, `%` `/`, `//`, `**`, `=`, `+=`, `-=`, `*=`, `/=`, `//=` ;
– les commandes en terminal `ls` et `cd`.
Lisez le *Mode d'emploi introductif pour les salles du NO4 et NO3* partie 1 « Introduction » et la partie 2 « La console Linux » (seulement 2.1, commandes `ls` et `cd`).

Exercices en séance

Ex. 1.1. Allez dans le menu Applications et lancez l'application *Terminal* située dans l'onglet Accessoires. Lancez *Python 3* en tapant la commande `python3` dans le Terminal.

Ex. 1.2. Entrez les expressions suivantes dans l'interpréteur et regardez le résultat :

1. `5`
2. `5 + 1`
3. `x = 5`
4. `x + 1`
5. `x = x + 1`
6. `x`

Ex. 1.3. Utilisez l'interpréteur pour afficher les textes suivants (rappel : `print()`) :

1. `Hello World`
2. `Aujourd'hui`
3. `C'est "Dommage !"`
4. `Hum \0/`

Allez dans le menu Applications et lancez l'application *pluma* (Éditeur de texte) située dans l'onglet Accessoires (alternativement, tapez « `pluma&` » dans le terminal). Créez un fichier avec les commandes que vous avez tapées pour les exercices précédents dans le terminal. Lancez encore un terminal et exécutez le programme que vous venez d'écrire a l'aide de la commande

`python3 nom_de_fichier.py` (positionnez-vous d'abord dans le bon dossier à l'aide des commandes `cd` et `ls`).
Exemple d'un tel fichier :

```
# Mon fichier Python
commande1
commande2
...
commandeN
```

Ex. 1.4. Évaluez (à la main) les expressions suivantes et essayez de deviner le type du résultat. Utilisez ensuite l'interpréteur Python pour vérifier vos réponses (rappel : `type()`). Résolvez les exercices avec divisions en utilisant l'opérateur `/`, puis l'opérateur `//`.

1. 14 - 14
2. 1 + 6.9
3. 1.0 + 2.0
4. $\frac{18}{7+1}$
5. $\frac{(3+2)*2.5}{4*2}$
6. 0 * 0.0
7. $4^{0.5}$
8. $\frac{5}{8}$
9. $-\frac{1}{2}$
10. $3^{-\frac{1}{2}}$

Ex. 1.5. Certaines des lignes de code suivantes contiennent des erreurs. Il peut s'agir d'erreurs syntaxiques ou sémantiques et certaines lignes génèrent des exceptions. Indiquez pour chacune d'entre elles le type d'erreur (s'il y en a) ou le résultat et expliquez brièvement. Vérifiez ensuite à l'aide de l'interpréteur. Le résultat désiré par le programmeur est indiqué en gras.

1. `print (''Bonjour')`
   `Bonjour`
2. `'bla' * 3.0`
   `'blablabla'`
3. `((1 + 4) / (6 * 2)`
   `0.4166666666666667`
4. `int(''14'')`
   `14`
5. `int('3+4')`
   `7`
6. `'3 * 3' * 3 ** 2`
   `81`
7. `3 + 2 / 0 + 2`
   `2`
8. `(print 'Il y a 31 jours en janvier')`
   `Il y a 31 jours en janvier`

9. `print ("Chuck Norris was born in " + 1940)`
   `Chuck Norris was born in 1940`

Ex. 1.6. Résolvez les problèmes suivants en écrivant des petits programmes dans des fichiers séparés. Créez d'abord toutes les variables nécessaires, tapez ensuite la formule en une seul ligne et affichez le résultat :

1. Le volume d'une sphère de rayon r est donné par $\frac{4}{3}\pi r^3$. Quel est le volume d'une sphère de rayon 5 ? Et de rayon 8 ?
2. Le prix affiché d'un livre est de 24.95 €, mais vous bénéficiez d'une réduction de 40 %. Par ailleurs, les frais d'envoi sont de 3 €. Quel est le prix total pour 60 livres ? Quel est le prix total de 50 livres si les frais d'envoi sont de 5 € et que vous bénéficiez d'une réduction de 43 % ?
3. Si vous parcourez 10 kilomètres en 43 minutes et 30 secondes, quelle est votre vitesse moyenne en miles par heure ? Quelle est votre vitesse moyenne en miles par heure si vous parcourez 10 kilomètres en 45 minutes ? Pour rappel : 1,61 km = 1 mile, 1 heure = 60 minutes et 1 minute = 60 secondes.
4. L'édition complète de la série « *Les comptes de Chuck Norris* » est composée de 3486 volumes et se trouve dans ses armoires numérotées dans l'ordre. Si chaque armoire peut contenir au plus 89 livres, dans laquelle se trouve le volume numéro 1024 ? Dans quelle armoire se trouverait le volume 404 si chacune pouvait contenir 91 livres ?

Ex. 1.7. Supposez que vous ayez quatre variables : a, b, c et d. Comment pourriez-vous vous y prendre pour inverser l'ordre des valeurs qu'elles réfèrent sans utiliser l'assignation multiple comme `a,b,c,d = d,c,b,a` ? Par exemple, si à l'initialisation, a = 1, b = "2", c = 3.0 et d = True, comment obtenir `"True 3.0 2 1"` à l'écran en entrant `print(a,b,c,d)` ?

Ex. 1.8. ** Même question que dans l'exercice 1.7 en supposant que les 4 variables sont de type `int`. Vous ne pouvez pas utiliser d'autres variables que `a`, `b`, `c` et `d`.

Ex. 1.9. Allez sur le site Python.org et cherchez dans la documentation du langage (« language reference ») de l'aide à propos de `print`. Comparez avec le résultat obtenu dans l'interpréteur lorsque vous entrez `help('print')`. Cherchez à vous renseigner sur d'autres aspects de Python vus au cours (`int`, `str`, les opérateurs, ...).

Chapitre 2

Contrôle de flux

Exercices préparatoires

Matière à réviser :
– les variables ;
– l'indentation ;
– les fonctions prédéfinies `input()`, `print()`, `range()` ;
– les types `int`, `float`, `bool` ;
– les opérateurs de comparaison et connecteurs logiques (>, <, ==, `!=`, `and`, `or`, etc.) ;
– les instructions de branchement conditionnel `if-else`, `if-elif-else` ;
– les boucles `for` et `while`.
Faites tous les exercices préparatoires sur papier et ensuite vérifiez vos réponses à l'aide de l'interpréteur `python3`.

Prép. 2.1. Qu'écrit sur *output* le programme suivant quand on lui fournit en *input* les valeurs 2 et 6 ?

```
a = int(input())
a *= 2
b = int(input())
b += a
print(a)
print(b)
```

Prép. 2.2. Qu'écrit sur *output* le programme suivant quand on lui fournit en *input* les valeurs 2, 6 et 4 ?

```
a = int(input())
b = int(input())
a = b
b = int(input())
b += a
print(a,b)
```

Prép. 2.3. Qu'écrit sur *output* le programme suivant quand on lui fournit en *input* les valeurs 2 et 6 ?

```
b = int(input())
a = int(input())
a = b+1
```

```
print(a)
a = b+1
print(a)
a += 1
print(a,a+1)
```

Prép. 2.4. Qu'écrit sur *output* le programme suivant quand on lui fournit en *input* :

1. les valeurs 2 et 6 ?
2. les valeurs 8 et 3 ?
3. les valeurs 3 et 3 ?

```
a = int(input())
b = int(input())
if a > b:
  print(a)
  a = b
print(a)
```

Prép. 2.5. Qu'écrit sur *output* le programme suivant quand on lui fournit en *input* :

1. les valeurs 2 et 6 ?
2. les valeurs 8 et 3 ?
3. les valeurs 3 et 3 ?

```
a = int(input())
b = int(input())
if a > b:
  print(a)
  a = b
  print(a)
```

Prép. 2.6. Donnez la valeur des variables `a`, `b`, `c`, `arret`, `test1`, `test2` et `test3` après chacune des instructions ci-dessous :

```
a = 2
b = 3
c = 4
test1 = True
test2 = (b>=a) and (c>=b)
test3 = test1 or test2
arret = test3 and (not test2)
a += 1
b -= 1
c -= 2
test1 = True
test2 = (b>=a) and (c>=b)
test3 = test1 or test2
arret = arret or test2
```

Prép. 2.7. Qu'écrit sur *output* le programme suivant quand on lui fournit en *input* :

1. les valeurs 1 et 6 ?

2. les valeurs -8 et 2.5 ?

```
a = float(input())
b = float(input())
test = (a >= b)
if test or a < 0:
  print("Oui!")
else:
  print("Non!")
```

Prép. 2.8. Que fait cette suite d'instructions ?

```
a = 1
while a <= 5:
  print(a, end=' ')
  a += 1
print(a)
```

Prép. 2.9. Que fait cette suite d'instructions ?

```
a = 1
while a < 5:
  a += 1
  print(a, end=' ')
print(a)
```

Prép. 2.10. Que fait cette suite d'instructions ?

```
i = 0
for j in range(10):
  i += j
print(i)
```

Prép. 2.11. Quelle est la différence entre les quatre instructions suivantes ? Expliquez ce qu'elles font et essayez de les utiliser avec une boucle `for`.

```
range(42)
range(42,69)
range(42,69,3)
range(42,-69,-1)
```

Exercices en séance

Ex. 2.1. Écrire un programme qui affiche a^{17} (a lu sur *input*) en employant le moins de multiplications possibles, sans utiliser l'opérateur $\star\star$ ni de boucle.

Ex. 2.2. Écrire un programme qui affiche $18 \times a$ (a lu sur *input*) en employant le moins d'instructions possibles, sans utiliser l'opérateur $\star$ ni de boucle.

Branchements conditionnels

Ex. 2.3. Écrire un programme qui lit 3 nombres, et qui, si au moins deux d'entre eux ont la même valeur, imprime cette valeur (le programme n'imprime rien dans le cas contraire).

Ex. 2.4. Pour chacune des 4 instructions d'affichage (`print`), donner l'ensemble des valeurs de a pour lesquelles celles-ci seront exécutées.

```
a = int(input())
if a > 0:
  if a > 1:
    if a > 2:
      print(a-2)
    else:
      print(a-1)
  else:
    print(a)
else:
  print("Erreur")
```

Ex. 2.5. Écrire le morceau de code qui si, a est supérieur à 0, teste si a vaut 1, auquel cas il imprime « a vaut 1 » et qui, si a n'est pas supérieur à 0, imprime « a est inférieur ou égal à 0 ».

Ex. 2.6. Écrire le programme qui lit en input trois entiers a, b et c. Si l'entier c est égal à 1, alors le programme affiche sur output la valeur de $a + b$; si c vaut 2 alors le programme affiche la valeur de $a - b$; si c est égal à 3 alors l'output sera la valeur de $a \times b$. Enfin, si la valeur 4 est assignée à c, alors le programme affiche la valeur de $a^2 + b \times a$. Si c contient une autre valeur, le programme affiche un message d'erreur.

Ex. 2.7. Écrire un programme qui imprime la moyenne arithmétique ($\frac{a+b}{2}$) de deux nombres lus sur input.

Ex. 2.8. Écrire un programme qui imprime la moyenne géométrique ($\sqrt{ab}$) de deux nombres lus sur input.

Ex. 2.9. Écrire un programme qui lit deux nombres a et b sur input et qui calcule et affiche le nombre c tel que b soit la moyenne arithmétique de a et c.

Boucles simples

Ex. 2.10. Écrire l'équivalent des instructions suivantes sans le mot clé break :

```
i=0
while i < 10:
  if i >5:
    break;
  print(i)
  i=i+1
```

Ex. 2.11. Écrire l'équivalent des instructions suivantes sans le mot clé pass :

```
i=0
while i < 10:
  if i == 5:
    pass
  elif i == 9:
    pass
  pass
  print(i)
  i=i+1
```

Ex. 2.12. Écrire l'équivalent des instructions suivantes sans le mot clé continue :

```
for i in range(11):
  if i%2 == 0:
    print(str(i)+" est pair")
    continue
  print(str(i)+" est impair")
```

Ex. 2.13. Écrire l'équivalent des instructions suivantes avec une boucle `while` :

```
for i in range(a,b+1):
  print(i)
```

Ex. 2.14. Remplacez la boucle `while` une boucle `for` dans le code ci-dessous :

```
i = 1
while i < 42:
  print(i)
  i += 1
```

Ex. 2.15. Pour cette exercice vous ne pouvez pas utiliser d'instruction `if`. Écrire un programme qui lit un nombre n sur input et qui affiche tous les nombres naturels :

1. entre 0 et n compris, de manière croissante (utilisez la boucle `for`) ;
2. entre 0 et n compris, de manière décroissante (utilisez la boucle `while`) ;
3. pairs entre 0 et n compris, de manière croissante (utilisez la boucle `while`) ;
4. entre 0 et n, bornes non comprises, de manière croissante (utilisez la boucle `for`) ;
5. multiples de 7 entre 0 et n bornes non compris, de manière croissante (utilisez la boucle `for`) ;
6. multiples de 5 entre 0 et n compris, de manière décroissante (utilisez la boucle `for`) ;

Ex. 2.16. Modifiez le code ci-dessous en utilisant une boucle `for` :

```
i = 1
while i < 42:
  print(i)
  i = 2*i
```

Ex. 2.17. Écrire un programme qui calcule la moyenne des résultats d'une interrogation, ces notes étant données en input, la fin des données étant signalée par la valeur sentinelle -1 (on suppose aussi que la suite des notes contient toujours au moins un élément).
Exemple d'exécution :

```
Entrez les notes.
Note 1 : 2
Note 2 : 8
Note 3 : 14
Note 4 : -1
La moyenne est de 8
```

Ex. 2.18. Écrivez un mini jeu : le programme choisit un nombre naturel (nombre secret) entre 0 et 100 de manière aléatoire, le jouer doit deviner ce nombre en utilisant le moins d'essais possibles. A chaque tour le joueur peut faire un essais et le programme doit répondre si le nombre

secret est plus grand ou plus petit. Si le joueur trouve le nombre secret, alors le programme doit afficher `"Bravo !"` ainsi que le nombre d'essais utilisés par le joueur.
Le joueur a maximum 7 essais ; s'il ne devine pas le secret après 7 essais, le programme s'arrête en écrivant `"Vous avez perdu !"`.
Pour générer des nombres aléatoires importez le module `random`. Pour générer un entier aléatoire dans intervalle $[0; n]$ utilisez la fonction `randint(0,n)`.
Exemple d'exécution :

```
J'ai choisi un nombre secret !
Quel est le secret ? 50
Non, le secret est plus petit !
Quel est le secret ? 25
Non, le secret est plus grand !
Quel est le secret ? 37
Bravo !
Le secret (37) est trouvé en 3 essais.
```

Ex. 2.19. * On peut calculer approximativement le sinus de x en effectuant la sommation des n premiers termes de la série (c'est-à-dire la somme infinie)

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots$$

où x est exprimé en radians.
Réécrivez cette somme sous la forme

$$\sin(x) = \sum_{i=0}^{\infty} f(i, x)$$

On vous demande donc de trouver $f(i, x)$. Écrivez ensuite le code calculant de cette manière la valeur de $\sin(x)$ où x est lu sur input. Continuez l'addition des termes successifs dans la série jusqu'à ce que la valeur d'un terme devienne inférieur (en valeur absolue) à une constante ε (par exemple, $\varepsilon = 10^{-5}$, ce qu'on noter `1e-5` en Python).

Boucles imbriquées

Ex. 2.20. Écrire un programme qui lit sur input une valeur naturelle n et qui affiche à l'écran un carré de n caractères `X` de côté, comme suit (pour $n = 6$) :

```
X X X X X X
X X X X X X
X X X X X X
X X X X X X
X X X X X X
X X X X X X
```

Ex. 2.21. Variante de l'exercice 2.20, afficher le triangle supérieur droit, comme suit (pour $n = 6$) :

```
X X X X X X
  X X X X X
    X X X X
      X X X
        X X
          X
```

D'autres variantes :
– afficher uniquement le bord du carré ;
– afficher le triangle inférieur gauche (supérieur gauche, inférieur droit) ;
– afficher un rectangle.

Ex. 2.22. Refaire l'exercice 2.20 en supposant que n est impair et en dessinant des `O` sur les deux diagonales principales à la place des `X`. Par exemple, pour $n = 7$:

```
O X X X X X O
X O X X X O X
X X O X O X X
X X X O X X X
X X O X O X X
X O X X X O X
O X X X X X O
```

Ex. 2.23. Refaire l'exercice 2.22 sans utiliser de branchement conditionnel (pas de `if-else`).

Chapitre 3

Fonctions

Exercices préparatoires

Pour les exercices sur le module « droite », veuillez revoir la géométrie euclidienne de base, en particulier la manipulation de droites de la forme analytique $y = ax + b$.

Prép. 3.1. Quel est l'effet des instructions suivantes ?

```
a, b = 1, 3.5
x, y = a, b
a, b = b, a
```

Prép. 3.2. Y a-t-il une différence sémantique entre les instructions suivantes ?

```
x,y = a,b
x,y = (a,b)
(x,y) = a,b
(x,y) = (a,b)
```

Prép. 3.3. Que vous donnent pour résultat les instructions suivantes ? Expliquez.

```
(1,2,3,4) <= (1,2,3,4,0)
(1,2,3,4) <= (1,1,8,4,-10)
(1,2,3,4) <= (1)
(1,2,3,4) <= (1,)
```

Prép. 3.4. Essayez l'instruction suivante :

```
a,b,c = input('Entrez trois chiffres')
```

avec les données suivantes comme entrées :
– 123
– 1 2 3
– abc
– 1b3
– 1

Prép. 3.5. Que fait le code suivant ?

```
t = tuple(range(10))
for x in t:
  print(x)
```

Prép. 3.6. En supposant que `x` et `y` sont des variables bien définies, la boucle suivante est-elle susceptible de changer leur valeur ? Expliquez.

```
for e in (x,y):
  e.foo()
  e = 4
```

Prép. 3.7. Le code suivant est-il correct ?

```
def plus2(x):
  """Renvoie la valeur donnée augmentée de 2."""
  return x+2 # On renvoie x+2
```

Prép. 3.8. Le code suivant est-il correct ?

```
def plus2(x):
  """Renvoie la valeur donnée augmentée de 2."""
  return x+2 """On renvoie x+2"""
```

Prép. 3.9. Entrez la fonction de l'exercice 3.7 dans l'interpréteur et entrez la commande `help(plus2)`. Expliquez ce qui se passe. Quid si vous tapez `help(bar)` ? Pourquoi y a-t-il une différence en tapant plutôt `help("bar")` ?

Prép. 3.10. Qu'affiche le code suivant ? Justifiez votre réponse.

```
def f(x):
  x = 3

x = 4
f(x)
print(x)
```

Prép. 3.11. Qu'aurait affiché le code de l'exercice 3.10 si la fonction `f` était plutôt :

```
def f(x):
  x = 3
  return x
```

Prép. 3.12. Qu'aurait-il fallu faire dans les exercices 3.10 et 3.11 pour que la variable `x` du code principal prenne la valeur 3 après l'appel à cette dernière fonction `f` ?

Prép. 3.13. Que fait la fonction suivante ?

```
def f(x,*args):
  print(len(args))
  print("x =", x)
  for arg in args:
    print(arg)
```

Essayez d'appeler cette fonction avec divers paramètres, par exemple `f()`, `f(4)`, `f(1,2,3,4,5)`.

Prép. 3.14. Le code suivant est-il syntaxiquement correct ? Quelle est sa sémantique ?

```
def f(x,y):
  return (x//y)/(x/y)

print(f(y=3,x=4))
```

Exercices en séance

Ex. 3.1. Écrivez une fonction `swap(a,b)` qui échange le contenu de `a` et `b` et renvoie `a` et `b` avec leur nouvelle valeur, afin de pouvoir l'utiliser comme suit : `a,b = swap(a,b)`.

Ex. 3.2. Écrivez une fonction `distancePoints()` qui, étant donnés deux points (x_1, y_1) et (x_2, y_2) reçus en paramètres sous forme de tuples, calcule et renvoie la distance euclidienne entre ces deux points.

Ex. 3.3. Écrivez une fonction `longueur(*points)` qui prend en paramètres un nombre arbitraire de points de coordonnées (x, y) et calcule la longueur du trait correspondant. Pour calculer la longueur, il faut effectuer la somme de la longueur des segments qui composent le trait. Un segment de droite est composé de deux points consécutifs passés en paramètre. Pour rappel, la distance entre deux points (x_1, y_1) et (x_2, y_2) se calcule comme suit :

$$\text{dist} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Ex. 3.4. Écrivez une fonction qui reçoit quatre points du plan en paramètres et calcule et affiche le périmètre du parallélogramme correspondant. Les points (x_1, y_1), (x_2, y_2), (x_3, y_3) et (x_4, y_4) correspondent au coin supérieur gauche, au coin supérieur droit, au coin inférieur droit et au coin inférieur gauche. La fonction n'affichera rien et renverra `False` si les côtés ne sont pas égaux deux à deux, elle renverra `True` sinon.

Ex. 3.5. Soit l'équation du second degré αx^2 + βx + $\gamma = 0$ où les paramètres α, β et γ sont entrés par l'utilisateur. Écrivez une fonction qui en calcule et affiche les solutions (s'il y en a).

Ex. 3.6. Dans le module `random`, la fonction `randint(a, b)` renvoie un nombre aléatoire compris entre a et b (inclus). Écrivez une fonction qui génère 3 nombres aléatoires représentant 3 dés à jouer (à six faces) et qui renvoie `True` si les dés forment un 421, `False` sinon.

Ex. 3.7. Considérons les billets et pièces de valeurs suivantes : 20 €, 10 €, 5 €, 2 €, 1 €. Écrivez une fonction `rendreMonnaie` qui prend en paramètres un entier *prix* et un 5-uple $(x_{20}, x_{10}, x_5, x_2, x_1)$ d'entiers représentant le nombre de billets et de pièces de chaque sorte que donne un client pour payer l'objet dont le prix est mentionné. La fonction doit renvoier un 5-uple représentant la somme qu'il faut rendre au client, décomposée en billets et pièces (dans le même ordre que précédemment). La décomposition doit être faite en utilisant le plus possible de billets et pièces de grosses valeurs.

Ex. 3.8. Écrivez une fonction `duree` qui prend deux paramètres `debut` et `fin`. Ces derniers sont des couples dont la première composante représente une heure et la seconde composante représente les minutes. Cette fonction doit calculer le nombre d'heures et de minutes qu'il faut pour passer de `debut` à `fin`.

Exemple : un appel à `duree((14, 39), (18, 45))` renvoie `(4, 6)`. Notez qu'un appel à `duree((6, 0), (5, 15))` renvoie `(23, 15)` et non `(0, 45)` !

Ex. 3.9. Même exercice mais sans utiliser les instructions conditionnelles.

Ex. 3.10. Écrivez une fonction `appliquer(a, b, f)` qui prend 2 entiers et une fonction `f(a,b)` en paramètre et qui renvoie le résultat de la fonction `f(a,b)` appliquée aux deux paramètres a et b si ce résultat est un entier, 0 sinon.

Ex. 3.11. Écrivez une fonction `sum(a, b)` qui prend 2 valeurs et renvoie la somme de a et de b. Par défaut, la valeur de a est 0 et la valeur de b est 1.

Ex. 3.12. Écrivez une fonction `sum(a, b)` en lambda calcul qui prend 2 entiers et renvoie la somme de a et de b.

Ex. 3.13. Écrivez une fonction `sum(a, b)` en lambda calcul qui prend 2 valeurs et renvoie la somme de a et de b si ces deux valeurs sont des entiers. Dans le cas contraire, la fonction renvoie 0. Par défaut, la valeur de a est 0 et la valeur de b est 1.

Ex. 3.14. Écrivez une fonction `nombreImpair()` qui utilise le mot clé `yield` afin d'afficher autant de nombres impairs que l'utilisateur y fait appel (grâce à la fonction `next`).

Ex. 3.15. Écrivez un generatuer `nombresPair(n)` (`n` est le nombre de nombres pairs qu'on veut générer) qu'on peut utiliser avec une boucle **`for`**. Par exemple :

```
for p in nombresPair(3):
  print(p)
```

Ex. 3.16.* Le n-ième nombre de Bell, noté B_n, est le nombre de partitions possibles d'un ensemble de n éléments. Il se calcule à partir de la formule suivante :

$$B_n = \sum_{k=1}^{n} \frac{1}{k!} \sum_{i=1}^{k} (-1)^{k-i} \frac{k!}{i!(k-i)!} i^n$$

On vous demande d'écrire la fonction `bell(n)` qui renvoie le n-ième nombre de Bell. Vous ne pouvez pas faire appel aux fonctions du module `math`. Nous demandons également que le code soit le plus efficace possible.

Module « droite »

Pour cet série d'exercices, nous allons créer un ensemble de fonctions permettant de manipuler des droites (non verticales) dans le plan euclidien. Nous allons placer ces fonctions dans un *module*. Veuillez créer un script (fichier) nommé `droite.py` et placez-y l'implémentation des fonctions décrites ci-dessous. Pour rappel, vos fonctions peuvent se servir des autres fonctions déjà implémentées.

Notations : Nous représentons une droite d dans le plan euclidien $\mathbb{R}^2$ par le tuple $(a, b) \in \mathbb{R}^2$ de telle manière à ce que :

$$d \equiv y = ax + b$$

De plus, nous représentons un point du plan euclidien $\mathbb{R}^2$ par un couple de réels.
Implémentez les fonctions suivantes en supposant que toutes les droites traitées sont telles que $a \neq 0$ (droites ni horizontales, ni verticales) :

Ex. 3.17. `droite(p1, p2)`
Entrée : Deux points distincts `p1` et `p2`.
Sortie : Un tuple (a,b) représentant la droite passant par `p1` et `p2`, `None` si les deux points ont la même abscisse.
Rappel : $a = \frac{y_2 - y_1}{x_2 - x_1}$ et $b = y_1 - ax_1 = y_2 - ax_2$.

Ex. 3.18. `appartient(d, p)`
Entrée : Une droite `d` et un point `p`.
Sortie : `True` si `p` $\in$ `d`, `False` sinon.
Rappel : vérifier si $y_p = ax_p + b$.

Ex. 3.19. `coefficientAngulaire(d)`
Entrée : Une droite `d`.
Sortie : Le coefficient angulaire de la droite `d`.
Rappel : a est le coefficient angulaire d'une droite $y = ax + b$.

Ex. 3.20. `intersectionAbscisses(d)`
Entrée : Une droite `d`.
Sortie : Le point intersection de `d` avec l'axe des abscisses.
Rappel : $(-\frac{b}{a}, 0)$.

Ex. 3.21. `paralleles(d1, d2)`
Entrée : Deux droites `d1` et `d2`.
Sortie : `True` si `d1` et `d2` sont parallèles, `False` sinon.
Rappel : vérifier si les coefficients angulaires sont identiques.

Ex. 3.22. `confondues(d1, d2)`
Entrée : Deux droites `d1` et `d2`.
Sortie : `True` si `d1` et `d2` sont confondues, `False` sinon.
Rappel : vérifier si les coefficients angulaires et les termes indépendants sont indentiques.

Ex. 3.23. `intersection(d1, d2)`
Entrée : Deux droites `d1` et `d2`.
Sortie : Un point d'intersection de `d1` et `d2` s'il existe, `None` sinon.
Rappel : Si les droites sont confondues, choisir un point quelconque d'une des droites. Si elles ne sont pas confondues mais parallèles, alors il n'y a pas d'intersection. Sinon, l'abscisse de l'intersection est donnée par $x_\cap = \frac{b_2 - b_1}{a_1 - a_2}$ et l'ordonnée par $y_\cap = a_1 x_\cap + b_1 = a_2 x_\cap + b_2$.

Ex. 3.24. `droiteNormale(d, p)`
Entrée : Une droite `d` et un point `p`.
Sortie : La droite perpendiculaire à `d` passant par `p`.
Rappel : $y = -\frac{x}{a} + (y_p + \frac{x_p}{a})$.

Ex. 3.25. `droiteParallele(d, p)`
Entrée : Une droite `d` et un point `p`.
Sortie : La droite parallèle à `d` passant par `p`.
Rappel : $y = ax + (y_p - ax_p)$.

Ex. 3.26. `distanceDroite(d, p)`
Entrée : Une droite `d` et un point `p`.
Sortie : La distance entre `d` et `p`.
Rappel : distance euclidienne entre p et le point p' qui est l'intersection entre la droite d et la droite normale à d passant par p.

Ex. 3.27. `symetrieOrthogonale(d, p)`
Entrée : Une droite `d` et un point `p`.
Sortie : Le point qui est l'image de `p` par la symétrie orthogonale d'axe `d`.
Rappel : Soit p' l'intersection entre la droite d et la droite perpendiculaire à d passant par p. Alors la symétrie orthogonale de p par d est le point $p'' = (x_p + 2(x_{p'} - x_p), y_p + 2(y_{p'} - y_p))$.

Ex. 3.28. ** **(pour les férus de maths)** Vous pouvez généraliser ces fonctions à des droites quelconques dans le plan euclidien, mêmes verticales, en supposant plutôt que les droites sont représentées par un triplet (a, b, c) de réels qui correspondent à une équation $ax + by + c = 0$. Écrivez dès lors les fonctions précédentes en tenant compte de la possibilité d'avoir des droites verticales et horizontales.

Chapitre 4

Chaînes de caractères

Exercices préparatoires

Prép. 4.1. Pourquoi les instructions suivantes ne donnent-elles pas le même résultat ?

```
"0199" < "187"
int("0199") < int("187")
```

Prép. 4.2. Que fait l'instruction suivante ?

```
"Tim" in "Castle Aaaaargh"
```

Prép. 4.3. Que fait le code suivant ?

```
txt = "Antioch"
for i in range(len(txt)):
  print(txt[i])
```

Prép. 4.4. Que fait le code suivant ?

```
for k in "Caerbannog":
  print(k,end=k)
```

Prép. 4.5. Expliquez pourquoi la seconde instruction du code suivant provoque une erreur.

```
msg = "Caerbannog"
msg[0] = "K"
```

Prép. 4.6. Quelle est la différence entre la chaîne vide `""` et `None` ?

Prép. 4.7. En supposant que le fichier `test.txt` existe et est bien accessible, que fait le code suivant ? Expliquez pourquoi on fait appel à la méthode `strip()`.

```
fd = open("test.txt")
for i in fd:
  print(i.strip())
fd.close()
```

Prép. 4.8. En supposant que le fichier `test.txt` existe et est bien accessible, que fait le code suivant ?

```
fd = open("test.txt")
s = fd.readlines()
fd.close()
for i in s:
  print(i.strip())
```

Prép. 4.9. En supposant que le fichier `test.txt` existe et est bien accessible, que fait le code suivant ?

```
fd = open("test.txt")
s = fd.read()
while s != '':
  print(s.strip())
  s = fd.read()
fd.close()
```

Prép. 4.10. Expliquez, en termes de consommation de mémoire, quelles méthodes d'accès de fichier parmi celles montrées aux préparations 4.7 à 4.9 sont efficaces et inefficaces.

Exercices en séance

Ex. 4.1. Écrivez les fonctions suivantes sous la forme d'un module appelé `stringmanip`. Dans tous les cas de figure, vos fonctions doivent admettre des indices positifs et négatifs.

1. Une fonction `caractere(s, i)` qui renvoie le i^e caractère de la chaîne de caractères `s`. Si ce caractère n'existe pas, la fonction renvoie la chaîne vide.
2. Une fonction `caracteres(s, i, j)` qui renvoie une chaîne de caractères contenant les caractères compris entre la position `i` et la position `j` (bornes incluses). Si ces bornes sont invalides, la fonction renvoie la chaîne vide.
3. Une fonction `change_caractere(s, i, a)` qui renvoie une chaîne de caractères identique à `s` dans laquelle le i^e caractère a été remplacé par le caractère stocké dans `a`. Si cette position est invalide ou que `a` contient plus d'un caractère, la fonction renvoie la chaîne vide.
4. Une fonction `change_caracteres(s, i, j, t)` qui renvoie une chaîne de caractères identique à `s` dans laquelle les caractères situés entre la position `i` et `j` (bornes incluses) ont été remplacés par la chaîne de caractères stockée dans `t`. Si les bornes `i` et `j` sont invalides ou que la chaîne `t` est trop courte ou trop longue, la fonction renvoie la chaîne vide.
5. Une fonction `trouve_caractere(s,a)` qui parcourra la chaîne `s` lettre par lettre et renverra la position (positive) de la première lettre identique à `a` rencontrée dans la chaîne. S'il n'y aucune lettre `a` dans la chaîne de caractère, la fonction renvoie -1.

Ex. 4.2. Écrivez les fonctions suivantes (prenant toutes une chaîne de caractères en paramètre) sous la forme d'un module appelé `userinput` :

1. `convert_to_int` qui, si la chaîne de caractères représente un nombre entier, effectue la conversion et renvoie l'entier correspondant. Si cette chaîne représente tout autre chose, la fonction affichera un message d'erreur.

   Conseil : utilisation de la méthode `isdigit()` appartenant au type chaîne de caractères.

2. `convert_to_float` qui, si la chaîne de caractères représente un nombre réel, renvoie le nombre réel. Si cette chaîne représente tout autre chose, la fonction affichera un message d'erreur. Vous ne pouvez pas utiliser la fonction `float()`.

   Conseil : Pour rappel, un nombre entier est aussi un nombre réel dont la partie décimale est égale à 0. Les autres nombres réels (qui ne sont pas entiers) peuvent s'écrire sous la forme *partie_entière.partie_décimale* (pensez à la méthode `split()` sur les chaînes de caractères).

3. `is_one_word` est une fonction qui prend une chaîne de caractères et qui, si celle-ci contient un seul mot, renvoie `True`. Sinon, la fonction renvoie `False`.

4. `is_one_letter` est une fonction prenant une chaîne de caractères en paramètre et qui renvoie `True` si la chaîne de caractères représente une seule lettre. Sinon, la fonction renvoie `False`.

5. `belongs_to_dictionary` est une fonction prenant en paramètre une chaîne de caractères et vérifie si elle représente un mot qui figure dans la liste des mots contenus dans le fichier `words.txt`, auquel cas elle renverra `True`. Sinon, la fonction renvoie `False`. Attention : les mots du fichier `words.txt` seront stockés ligne par ligne dans le fichier.

Ex. 4.3. Écrivez une fonction `borders` qui prend une chaîne de caractères en paramètre et renvoie une liste contenant les bords de la chaîne de caractères. On dit qu'un mot `u` est un bord de `w` (avec $u \neq w$) si `u` est à la fois un préfixe strict (c'est-à-dire non vide) de `w` et un suffixe strict (c'est-à-dire non vide) de `w`. Si `w` n'a pas de bord, la fonction renvoie la chaîne vide. Exemple : *'a'* et *'abda'* sont des bords de *'abdabda'*. Le plus grand bord est *'abda'*.

Ex. 4.4. Écrivez une fonction `plus_grand_bord(w)` qui, étant donné un mot `w`, renvoie le plus grand bord de ce mot.

Ex. 4.5. Écrivez une fonction `anagrammes(v, w)` qui renvoie `True` si et seulement si les mots `v` et `w` sont des anagrammes, c'est-à-dire des mots qui comprennent les mêmes lettres mais pas nécessairement dans le même ordre. Par exemple, *'marion'* et *'romina'* sont des anagrammes.

Ex. 4.6. Écrivez une fonction `intersection(v, w)` qui calcule l'intersection entre deux chaînes de caractères `v` et `w`. On définit l'intersection de deux mots comme étant la plus grande partie commune à ces deux mots. Par exemple, l'intersection de « *programme* » et « *grammaire* » est « *gramm* ».

Ex. 4.7. Ecrire une fonction `trans(s, (a_rep, a), (b_rep, b))`, qui reçoit trois paramètres, et renvoie le résultat de la transformation suivante : chaque occurrence du symbole `a_rep` dans la chaîne `s` est remplacée par la chaîne `a`, et chaque occurrence du symbole `b_rep` est remplacée par la chaîne `b`.
Exemple :

```
>>> print(trans('ABBAB', ('A','AB'), ('B','BA')))
>>> 'ABBABAABBA'
```

Ex. 4.8. La méthode de chiffrement par décalage est une des premières techniques utilisées pour chiffrer un message. On appelle aussi cette technique le « chiffre césarien ». Dans cette méthode, on décale chaque lettre du message de k unités. Par exemple, si $k = 3$, la lettre « A » devient « D » par chiffrement. De même, si l'on considère le mot « BONJOUR », nous obtenons avec un décalage de trois unités le mot « ERQMRXU ». Le décalage est circulaire : un décalage

Texte clair	Z	E	D		C	A	B
Clef	A	B	B	A	A	B	B
Texte chiffré	Z	F	E		C	B	C

FIGURE 4.1 – Chiffrement par l'algorithme de Vigenère.

de trois unités sur la lettre « X » a par exemple pour résultat la lettre « A ». Nous considérons pour cet exercice que les messages ne sont composés que de caractères alphabétiques majuscules et non accentués.
Pour effectuer ce genre de décalages, nous pouvons utiliser la correspondance entre lettres et leur valeur numérique dans un encodage particulier, par exemple l'ASCII. La Table 4.1 reprend la correspondance entre les majuscules de « A » à « Z » et leur valeur numérique dans l'encodage ASCII. Il est important de remarquer que les valeurs se suivent en séquence ; en d'autres termes, pour obtenir la valeur numérique correspondant à la lettre « D », il suffit d'ajouter 3 à la valeur numérique de la lettre « A ».

Lettre	**Valeur ASCII**
A	65
B	66
⋮	⋮
Y	89
Z	90

TABLE 4.1 – Correspondances entre majuscules et leur valeur dans l'encodage ASCII.

La fonction `ord()` en Python permet d'obtenir, pour une lettre donnée passée en paramètre, sa valeur numérique dans l'encodage ASCII. Par exemple, `ord('A')` renverra la valeur 65. La fonction `chr()` prend quant à elle une valeur numérique de l'encodage ASCII et renvoie la lettre correspondante : `chr(67)` renverra la lettre « C ».
À l'aide de ces deux fonctions, écrivez les fonctions suivantes :
– `caesar(mot, k)` qui renvoie le résultat du décalage de `k` unités du mot `mot`, reçu en paramètre (une chaîne de caractères).
– `canonique(mot)` qui renvoie une version canonique d'un mot. On définit la version canonique d'un mot comme étant l'unique décalage de ce mot donnant un mot débutant par la lettre « A ».
– Expliquez comment il serait possible de déchiffrer un message chiffré et quelle information devrait être transmise entre l'expéditeur et le destinataire pour que ce dernier comprenne bien le message.

Ex. 4.9.* (Examen de janvier 2012) Le *chiffre de Vigenère* est une version améliorée du chiffre césarien vu à l'exercice 4.8. Dans cet algorithme, la clef n'est plus un simple décalage dans l'alphabet mais est un texte à part entière de longueur arbitraire.
Pour chiffrer un texte clair à la main, on utilise la *table de Vigenère* dont un extrait est donné dans la Table 4.2. Supposons que la clef choisie soit « ABBA » et que nous voulions chiffrer le texte clair « ZED CAB ». Dans un premier temps, nous étendons la clef en la concaténant à elle-même jusqu'à ce qu'elle ait la même longueur que le texte clair (en tronquant les éventuels caractères superflus) ; la clef « ABBA » devient donc « ABBAABB » car le texte clair est de longueur 7. Si la clef était plus longue que le texte clair de longueur n, on la tronque pour ne garder que

	A	B	C	D	E	…	Z
A	A	B	C	D	E	…	Z
B	B	C	D	E	F	…	A
C	C	D	E	F	G	…	B
D	D	E	F	G	H	…	C
E	E	F	G	H	I	…	D
⋮	⋮	⋮	⋮	⋮	⋮	⋱	⋮
Z	Z	A	B	C	D	…	Y

TABLE 4.2 – Extrait de la table de Vigenère. Les lignes correspondent à une lettre du texte clair, les colonnes à une lettre de la clef, et les cases du tableau aux lettres correspondantes du texte chiffré.

les n premiers caractères. Ensuite, pour chaque caractère du texte clair, nous consultons la ligne correspondante dans la table, dans la colonne du caractère de la clef à la même position, et nous en tirons une lettre qui sera la lettre correspondante dans le chiffré. La première lettre du texte clair, « Z », restera donc « Z » dans le chiffré car chiffrer par la lettre « A » revient à ne rien changer si nous consultons le tableau. La seconde lettre du texte clair, « E », se fera chiffrer par la lettre « B » ; la table nous indique que le caractère dans le chiffré correspondant sera donc « F ». La Figure 4.1 vous donne le chiffrement complet de « ZED CAB » par la clef « ABBA ».
Nous vous demandons d'écrire une fonction `vigenere()` prenant deux paramètres : une chaîne de caractères `plaintext` ainsi qu'une chaîne de caractères `key`. Cette fonction devra lire et chiffrer le texte clair donné par `plaintext` en utilisant la clef fournie, et renvoyer le texte chiffré. Vous pouvez supposer que le texte clair ne contient que des lettres majuscules ainsi que des symboles de ponctuation et des espaces. Notez que si on rencontre autre chose que des lettres, on laisse le caractère intact dans le chiffré (voir comportement vis-à-vis d'un caractère espace en Figure 4.1).
Vous pouvez utiliser les fonctions `ord()` et `chr()` vues à l'exercice 4.8. Nous vous encourageons à écrire des fonctions supplémentaires si cela peut vous aider ou clarifier votre code. Nous vous demandons d'utiliser parcimonieusement la mémoire ; évitez par exemple si possible de construire la table de Vigenère en mémoire.

Chapitre 5

Listes

Exercices préparatoires

Prép. 5.1. Quelle est la différence entre un tuple et une liste ? Pour vous aidez, exécutez le code suivant :

```
a=(4,2,1)
a[1]=9
print(a[1])
b=[4,2,1]
b[1]=9
print(b[1])
```

Prép. 5.2. Que réalise chaque ligne du code suivant :

```
a = []
a.append(1)
print(a)
a.append(2)
print(a)
a.append([3])
print(a)
a=a+[4,5,6,7]
print(a)
print(a[1:3])
print(a[3:])
print(a[:2])
```

Prép. 5.3. **(Examen de janvier 2012)** Que donne le code suivant ?

```
x = [1]
x[1] = 2
print(x)
```

Prép. 5.4. **(Examen de janvier 2012)** Que donne le code suivant ?

```
def foo(v):
  v = [3,4]
x = [0,1,2]
foo(x)
print(x)
```

Prép. 5.5. **(Examen de janvier 2012)** Que donne le code suivant ?

```
x = [0,1,2]
y = x
y[:] = [3,4]
print(x,y)
```

Prép. 5.6. Construisez une liste L contenant un entier quelconque compris entre 1 et 10. Multipliez le par 2, ajoutez 8 et divisez le par 2. Soustrayez le nombre que vous avez choisi au départ à ce nouveau nombre. Verifiez que le résultat vaut 4.

Prép. 5.7. Expliquez pourquoi le programme suivant affiche deux fois le même résultat.

```
L1 = []
L2 = L1
L1.extend([1,2])
L2.extend([3,4])
print(L1, L2, sep='\n')
```

Prép. 5.8. Quel résultat affiche ce code ? Expliquez pourquoi.

```
L1 = []
L2 = L1
print(L1 == L2)
print(L1 is L2)
```

Prép. 5.9. Quel résultat affiche ce code ? Expliquez pourquoi.

```
L1 = []
L2 = []
print(L1 == L2)
print(L1 is L2)
```

Prép. 5.10. En supposant que `L1` et `L2` soient des listes vides distinctes, quel est le résultat des opérations suivantes ? Expliquez la différence entre les deux.

```
L1.append('hello')
L2.extend('hello')
```

Exercices en séance

Note : Dans les exercices qui suivent, nous vous demandons de ne pas utiliser les fonctions suivantes, prédéfinies par Python pour manipuler les listes : `insert`, `count`, `remove`, `map`, `filter`, `enumerate`.

Ex. 5.1. Écrivez une fonction `bornes(nombres)` qui reçoit en paramètre une liste de nombres et renvoie un tuple comprenant le maximum et le minimum des valeurs entières présentes dans cette liste.

Ex. 5.2. Écrivez une fonction `my_pow` qui prend comme paramètres un nombre entier `max` et un nombre flottant `b` et qui renverra une liste contenant les `max` premières puissances de `b`, c'est-à-dire une liste contenant les éléments allant de $\texttt{b}^0$ à $\texttt{b}^{\texttt{max}-1}$.

Ex. 5.3. Écrivez une fonction `prime_numbers` qui prend comme paramètre un nombre entier `nb` et qui renverra une liste contenant les `nb` premiers nombres premiers. Nous vous demandons de penser aux différents cas pouvant intervenir dans l'exécution de la fonction.
Pour rappel, un nombre premier b est un entier naturel qui admet que deux diviseurs distincts entiers et positifs : b et 1. En d'autres termes, b est premier si il n'existe pas d'entiers naturels c et d différents de b et de 1 tel que $b = c \times d$. Mathématiquement, on peut définir un nombre premier b comme étant un nombre tel que

$$(b > 2) \wedge \Big(\nexists a \cdot (1 < a < b) \wedge (b \bmod a = 0)\Big)$$

Ex. 5.4. Écrivez une fonction `my_insert` qui prend une liste triée `L` et un entier `n` en paramètres. La fonction devra insérer `n` dans la liste tout en la maintenant triée.

Ex. 5.5. Écrivez une fonction `my_count` qui prend une liste `L` et un élément `e` en paramètres. La fonction doit renvoyer le nombre de fois que l'élément `e` apparaît dans la liste. N'oubliez pas de tester tous les cas possibles.

Ex. 5.6. Écrivez une fonction `my_remove` qui prend une liste `L` et un élément `e` en paramètre et qui effacera la première apparition de l'élément `e` dans la liste. N'oubliez pas de tester tous les cas possibles.

Ex. 5.7. Écrivez une fonction `my_map` qui prend une liste `L` et une fonction `f` en paramètres et qui renverra une nouvelle liste où un élément à la i^{e} position contiendra la valeur de retour de la fonction `f` appliquée au i^{e} élément de la liste `L`. À nouveau, n'oubliez pas de tester tous les cas possibles.

Ex. 5.8. Écrivez une fonction `my_filter` qui prend une liste `L` et une fonction `f` en paramètres. Cette fonction renverra une nouvelle liste constituée des éléments de la liste `L` pour lesquels la fonction `f` renvoie `True`.

Ex. 5.9. Écrivez une fonction `my_enumerate` qui prend une liste `L` en paramètre. Cette fonction renverra une liste de tuples à deux composantes. Chaque tuple devra avoir en premier élément l'indice i et en deuxième élément le i^{e} élément de la liste `L`. Veuillez tester votre programme et repérer clairement ses limites d'utilisation.

Ex. 5.10. Écrivez une fonction `reduce` qui prend en paramètres une liste `L`, une fonction `f` (à deux paramètres) et un élément `e`. La fonction devra être initialement appliquée à l'élément `e` et au premier élément de la liste `L`. Ensuite, il faudra successivement appliquer la fonction `f` sur le résultat du précédent appel de fonction et l'élément suivant de la liste.
Exemple : `reduce(somme, [1,2,3,4] ,0)` renverra $((((0 + 1) + 2) + 3) + 4)$

Ex. 5.11. Écrivez une fonction `my_print` qui prend en paramètres une liste `L` et un tuple `separator` de taille 3. Les éléments de la liste devront être des nombres entiers. La fonction devra renvoyer une chaîne de caractères contenant dans l'ordre :

1. le premier élément de `separator` ;
2. chaque élément de la liste `l` en insérant, entre chaque élément, le deuxième élément de `separator` ;
3. le troisième élément de `separator`.

Exemple : `>>> my_print([1, 2, 3, 4], ('[', '->', ']'))` affichera
`[1->2->3->4]`

Ex. 5.12. Écrivez une fonction `my_invert` qui inverse (en place) l'ordre des éléments dans une liste qui lui est donnée en paramètre sans utiliser une autre structure telle qu'une autre liste. *Exemple :* `my_invert([1, 2, 3, 4])` renverra `[4,3,2,1]`

Ex. 5.13. On se donne une liste qui encode une séquence t. Chaque élément de cette liste est un tuple de deux éléments : le nombre de répétitions successives de l'élément x dans t, et l'élément x lui-même. Les éléments successifs répétés forment la séquence t.

Écrivez une fonction `decompresse`, qui reçoit une telle liste en paramètre et renvoie la séquence t sous forme d'une nouvelle liste.

Exemple :

```
>>> L = [(4, 1), (2, 2), (2, 'test'), (3, 3), (1, 'bonjour')]
>>> decompresse (L)
[1, 1, 1, 1, 2, 2, 'test', 'test', 3, 3, 3, 'bonjour']
```

Chapitre 6

Matrices

Exercices préparatoires

Prép. 6.1. Écrivez une fonction `init_mat(m,n)` et une fonction `print_mat(M)`. La première construit une matrice d'entiers initialisée à la matrice nulle et de dimension $m \times n$. La seconde prend une matrice en paramètre et affiche son contenu.

Prép. 6.2. Qu'affiche `print_mat(M)` lorsqu'on exécute au préalable le code suivant :

```
L = [0,0]
M = [L,L]
L[0] = 2
```

Prép. 6.3. Le code précédent montre que la matrice `M` contient la liste `L`. Par conséquent, toute modification de `L` modifie l'affichage réalisé par `print_mat(M)`. Écrivez une fonction `my_deepcopy(L)` qui renvoie une nouvelle liste contenant les mêmes valeurs que `L`. Cette fonction vous permettra d'inclure les différentes valeurs de `L` (et non d'inclure `L`) à la matrice `M`.

Exercices en séance

Ex. 6.1. Écrivez une fonction `trace(M)` qui prend en paramètre une matrice `M` de taille $n \times n$ et qui calcule sa trace de la manière suivante :

$$\mathrm{tr}(A) = \sum_{i=1}^{n} A_{ii}$$

Ex. 6.2. Une matrice $M = \{m_{ij}\}$ de taille $n \times n$ est dite `antisymétrique` lorsque pour toute paire d'indices i, j, on a $m_{ij} = -m_{ji}$. Ecrire une fonction booléenne qui teste si une matrice est antisymétrique.

Ex. 6.3. Écrivez une fonction `produitMatriciel(A,B)` qui calcule le produit matriciel C (de taille $m \times \ell$) entre les matrices A (de taille $m \times n$) et B (de taille $n \times \ell$). Le produit matriciel se calcule comme suit :

$$C_{ij} = \sum_{k=0}^{n-1} A_{ik} B_{kj}$$

Ex. 6.4. Écrivez une fonction `xorMatriciel(A,B)` qui prend en paramètres deux matrices A et B de même dimensions (de taille $m \times n$), et qui renvoie une matrice résultant C contenant les éléments de A xorés au éléments de B. Cette fonction opère comme suit :

$$C_{ij} = A_{ij} \oplus B_{ij}, \forall i,\ j : 0 \leqslant i < n\ et\ \leqslant j < m$$

Ex. 6.5. Écrivez une fonction `rotationGauche(A)` qui prend en paramètre une matrice A sur laquelle elle effectue des rotations gauche sur chaque ligne. Le nombre de déplacements effectués par la rotation est en fonction du nombre de la ligne en question ; Autrement dit, la ligne i subit une rotation de i positions.
Comment adapteriez-vous cette fonction pour effectuer une rotation droite ?

Ex. 6.6.* (Examen de juin 2012) Soit une matrice A de taille $m \times n \times q$ contenant des entiers quelconques et une matrice P de même taille contenant des entiers compris entre 0 et $m \times n \times q - 1$, tous distincts. On vous demande de réorganiser la matrice A de la manière suivante : l'élément à l'indice (a, b, c) de la matrice A d'origine devra, après réorganisation, se trouver à la *position* numéro $P[a][b][c]$ de la matrice A. La matrice P est appelée *matrice de permutation* des éléments de la matrice A. Chaque élément de la matrice P à *l'indice* (a, b, c) contient la *position* où se trouvera l'élément (a, b, c) de A après l'ensemble des permutations.
On définit la *position* d'un élément comme l'*indice* qu'aurait cet élément dans un vecteur qui serait constitué des lignes de A mises côte-à-côte. Plus précisément, la Figure 6.1 vous montre les positions des différents éléments se trouvant dans une matrice A de dimension $3\times3\times2$. On y voit par exemple que l'élément $A[0][2][1]$ (c'est-à-dire première ligne, troisième colonne, deuxième niveau de profondeur) contient l'entier 11 et est donc en position 11. L'élément $A[1][1][0]$ quant à lui est à la position 4.

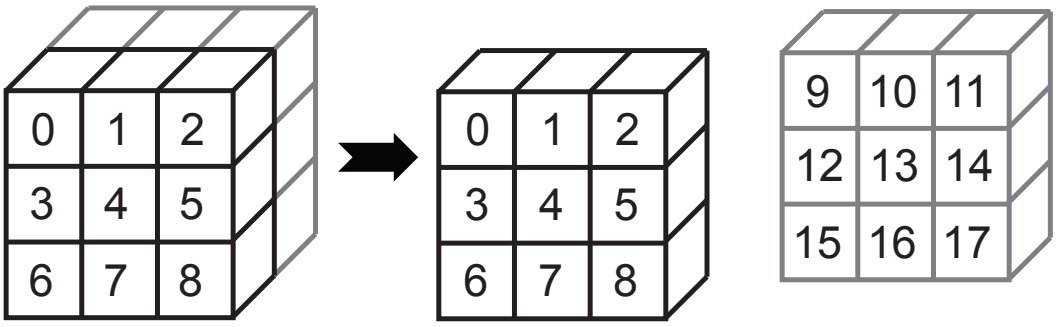
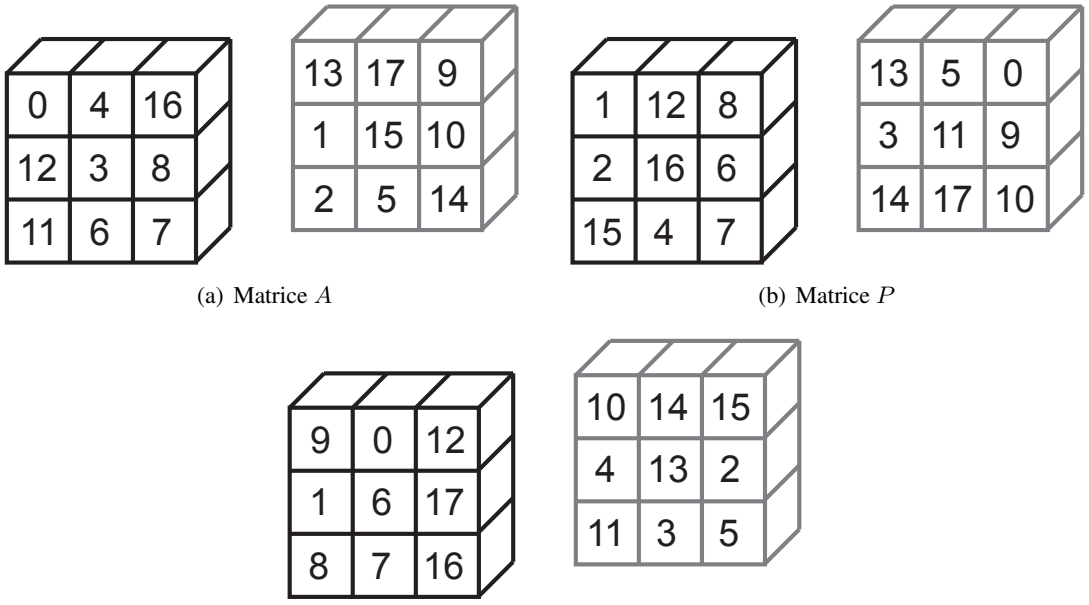


FIGURE 6.1 – Positions des différents éléments dans une matrice cubique.

On vous demande d'écrire un programme en Python qui permutera les éléments d'une matrice A en fonction d'une matrice de permutation P. Le traitement doit être effectué sans utiliser de structure intermédiaire (c'est-à-dire pas de dictionnaire, pas de nouvelle liste ou de matrice, etc.). En d'autres mots, vous avez le droit de modifier la matrice A, la matrice P ainsi que d'utiliser des variables ne contenant que des entiers.
La Figure 6.2 vous montre un exemple de matrice A, de matrice P et enfin d'une matrice A' qui est la matrice A après permutation des éléments grâce à la matrice P. On y voit que l'entier 9 en position $(0, 2, 1)$ dans la matrice A (i.e. A[0][2][1]) doit se trouver en position 0 dans la matrice A après réorganisation (donc en A[0][0][0]).
La Figure 6.3 vous affiche un autre exemple de matrice A, d'une matrice P et enfin d'une matrice A' qui est la matrice A après permutation des éléments grâce à la matrice P. Les flèches



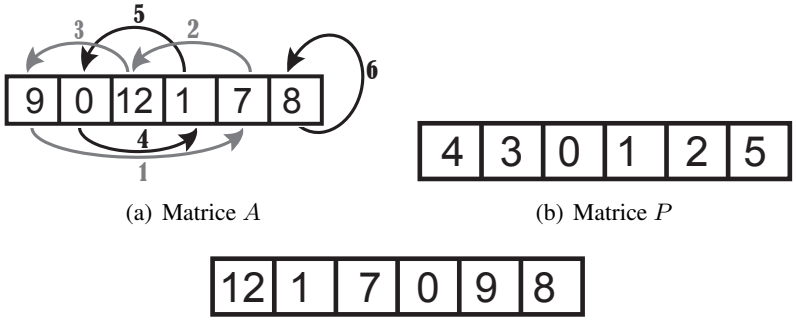
(a) Matrice A (b) Matrice P

(c) Matrice A'

FIGURE 6.2 – Exemple de matrice A (figure a), de matrice P (figure b) et de matrice A' (figure c) représentant l'application de la permutation P sur A.

mises dans la Figure 6.3(a) vous permettent de voir comment sont permutées les valeurs dans la matrice A. On y voit que :

1. la valeur de A à l'indice 0 est placée dans la case de A' à l'indice 4,
2. puis la valeur de A à l'indice 4 est placée dans la case de A' à l'indice 2,
3. puis la valeur de A à l'indice 2 est placée dans la case de A' à l'indice 0,
4. puis la valeur de A à l'indice 1 est placée dans la case de A' à l'indice 3,
5. puis la valeur de A à l'indice 3 est placée dans la case de A' à l'indice 1,
6. et enfin la valeur de la dernière case n'est pas déplacée.



(a) Matrice A (b) Matrice P

(c) Matrice A'

FIGURE 6.3 – Exemple de matrice A (figure a), de matrice P (figure b) et de matrice A' (figure c) représentant l'application de la permutation P sur A.

Ex. 6.7.* Écrivez une fonction `rotation(M)` qui effectue une rotation, en place, de +90° (dans le sens trigonométrique) d'une matrice carrée `M` de taille $n \times n$.

Ex. 6.8. **(mini-projet)** Le jeu du Morpion est un jeu dans lequel s'opposent deux joueurs. Le but dans ce jeu est d'aligner 5 fois un même jeton dans la grille de taille 5×5 de départ. Il vous est demandé d'implémenter une version du Morpion en suivant les consignes suivantes :

- Créez une fonction `create_morpion(n)` qui crée et renvoie une grille de taille $n \times n$ sur laquelle les joueurs seront amenés à jouer.
- Créez une fonction `next_turn(morpion, player, i, j)` qui simule l'ajout d'un jeton en position i, j par le joueur `player` (1 ou 2). Cette fonction renvoie une nouvelle grille correspondant à l'ajout du jeton sur la grille morpion passée en paramètre. Si le coup joué n'est pas valide (mauvaises positions, position déjà complète, etc.), cette fonction renvoie explicitement `None`.
- Créez une fonction `finished(morpion,i,j)` qui renvoie -1 si la partie n'est pas terminée, 0 si la grille correspond à une situation de match nul, 1 si le joueur ayant placé le dernier pion a gagné la partie. Un joueur emporte la partie s'il y a une ligne, une colonne ou une (des deux) diagonale qui ne contient que des jetons de ce joueur.
- Créez une fonction `play_morpion()` qui demande à l'utilisateur la taille de la grille, et propose à deux joueurs d'effectuer une partie du Morpion.

Chapitre 7

Dictionnaires

Exercices préparatoires

Matière à réviser :
– syntaxe du dictionnaire (`d[clef] = valeur`);
– fonctions `len()`, `values()`, `keys()`, `items()`, `clear()`, `copy()`, `get()`;
– opérateur `in`;
– l'exercice 4.8 (p. 19).

Prép. 7.1. Parmi les instructions suivantes, quelles sont celles qui engendrent une erreur ? Sans ces instructions erronées, que contient le dictionnaire `d` à la fin ?

```
d = {}
d[1] = 22
d[3.14] = 24
b = 4.5
d[b] = 90
d['b'] = 25
d[(1,2)]'X'
d[[3,4]] = 32
d[2] = (3,4)
d[5,4] = [5,4]
```

Prép. 7.2. Devinez le résultat de l'opération suivante :

```
sum( set( [1,1,1,2,1,1,2,3,5,8] ) )
```

Expliquez.

Exercices en séance

Ex. 7.1. Dans un texte, il nous arrive souvent de remplacer des mots par des abbréviations (exemple : « bonjour » par « bjr »). Nous allons utiliser un dictionnaire pour contenir les abbréviations et leur signification.
Nous vous demandons d'écrire une fonction `remplacer(message,abbreviation)` qui va créer une copie de la chaîne de caractères `message` dans laquelle les mots qui figurent dans le dictionnaire `abbreviation` (comme clé) sont remplacés par leur signification (valeur). Exemple d'utilisation :

```
abbrev = {'C.' : 'Chuck',
          'N.' : 'Norris',
          'cpt' : 'counted',
          '2' : '2 times',
          'inf' : 'infinity'}
print(remplacer('C. N. cpt 2 to inf', abbrev))
```

Ex. 7.2. Écrivez trois fonctions :
– `addition(a,b)` : qui renvoie la somme de `a` et `b`
– `soustraction(a,b)` : qui renvoie la différence entre `a` et `b`
– `multiplication(a,b)` : qui renvoie le produit de `a` et `b`
Nous vous demandons aussi de rédiger un script permettant à l'utilisateur d'agir avec ces fonctions selon son désir :

```
Rentrer deux valeurs :
a : 5
b : 6
Choississez :
- (A)dditionner a et b
- (M)ultiplier a par b
- (S)oustraire a à b
- (Q)uitter
```

Suivant la lettre entrée au clavier par l'utilisateur pour son choix (clé), on effectuera alors l'appel de la fonction correspondante en la sélectionnant dans un dictionnaire de fonctions (valeurs).

Ex. 7.3. Reprenez l'exercice 4.8 (p. 19) et écrivez une fonction `equiv_canonique()` prenant un nombre arbitraire de mots en paramètres. Votre fonction devra renvoyer un dictionnaire où chaque clef est une forme canonique et où la valeur correspondante est la sous-liste des mots passés en paramètres respectant cette forme. Par exemple,
`equivalence_canonique('BBB', 'CCC', 'BABA', 'ABBA')`
devra renvoyer un dictionnaire de la forme :
`{'AAA': ['BBB', 'CCC'], 'AZAZ': ['BABA'], 'ABBA': ['ABBA']}`.

Ex. 7.4. Écrivez une fonction `values(dico)` qui doit fournir, à partir du dictionnaire donné en paramètre, une liste des valeurs du dictionnaire telles qu'elles sont triées sur base de la clef du dictionnaire. Vous pouvez supposer que les clefs du dictionnaire sont des strings et vous pouvez utiliser la méthode `sort()` sur les listes pour vous aider.

Ex. 7.5. Écrivez une fonction `primesOddsNumbers(numbers)` qui reçoit une liste de nombres et qui affiche les nombres impaires de cette liste ainsi que les nombre premiers. Votre fonction doit faire appel à deux autres fonctions (que vous ne devrez pas écrire) qui renvoient, respectivement, un ensemble de nombres premiers et un ensemble de nombre paires.

Ex. 7.6. Écrivez une fonction `store_email(liste_mails)` qui reçoit en paramètre une liste d'adresses e-mail et qui renvoie un dictionnaire avec comme clefs le domaine des adresses e-mail et comme valeurs les listes d'utilisateurs correspondantes. Par exemple, la liste d'adresses suivante :

```
[ "jvervier@lit.ulb", "andre.colon@stud.ulb", "thierry@profs.ulb",
  "info@lit.ulb", "eric.ramzi@stud.ucl", "bernard@profs.ulb",
  "jean@profs.ulb" ]
```

donnerait le dictionnaire suivant :

```
{ "lit.ulb" : ["jvervier", "info"]
  "stud.ulb" : ["andre.colon"]
  "profs.ulb" : ["thierry", "bernard", "jean"]
  "stud.ucl" : ["eric.ramzi"] }
```

Ex. 7.7. **(mini-projet)** Un dictionnaire peut nous permettre de stocker un tableau partiel, c'est-à-dire uniquement les cases remplies du tableau avec comme valeur le contenu de la case en question. Voici un exemple :

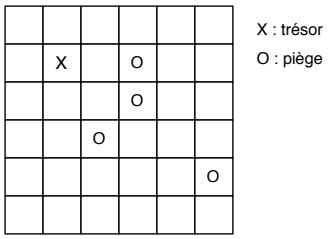


FIGURE 7.1 – Exemple de tableau

La tableau ci-dessus peut être implémenté de la manière suivante :

```
tresor = {}
tresor[(1,1)] = "tresor"
pieges = [(2,3),(4,5),(1,3),(3,2)]

for piege in pieges
    tresor[piege] = "piege"
```

On vous demande d'implémenter les fonctions suivantes :

- `creer_carte(n, nb_pieges)` qui reçoit en paramètres deux nombres naturels `n` et `nb_pieges` supérieurs ou égaux à 2 et qui renvoie un dictionnaire représentant une carte de taille `n` × `n` dans laquelle on place `nb_pieges` pièges et un trésor de manière aléatoire (utilisez le module `random`).
- `recherche_tresor(n, nb_pieges)` qui crée une carte de taille `n` × `n` dans laquelle `nb_pieges` pièges et un trésor ont été placés de manière aléatoire et qui demande à l'utilisateur d'entrer une coordonnée. La fonction doit demander à répétition des coordonnées jusqu'à ce que l'utilisateur trouve le trésor (gagné) ou tombe sur un piège (perdu).

Ex. 7.8.* **(Examen de juin 2011)** Dans les sciences expérimentales ainsi que dans bien d'autres domaines, on ne peut pas simplement se contenter d'effectuer des mesures mais il est également important de pouvoir caractériser celles-ci, comme par exemple via le calcul de moyenne, médiane, variance, etc. Un outil particulièrement utile dans ce contexte est *l'histogramme* dont le rôle est de pouvoir étudier la distribution de valeurs récoltées. Sa représentation la plus typique est sous forme d'un graphique en bâtons (voir Figure 7.2).
Étant donnée une liste L de valeurs (supposées réelles), nous pouvons représenter un histogramme par un dictionnaire qui associe, pour des gammes de valeurs données, le nombre de valeurs présentes dans L et ayant une valeur comprise dans cette gamme. Une application particulière d'un histogramme serait par exemple d'étudier les points d'étudiants à un examen donné ; l'histogramme pourrait alors décrire combien d'étudiants ont eu une note entre 0 et 4, entre 5 et 9, entre 10 et 14 et entre 15 et 20 par exemple. Notez que le nombre de catégories (c'est-à-dire de gammes de valeurs) est arbitraire.

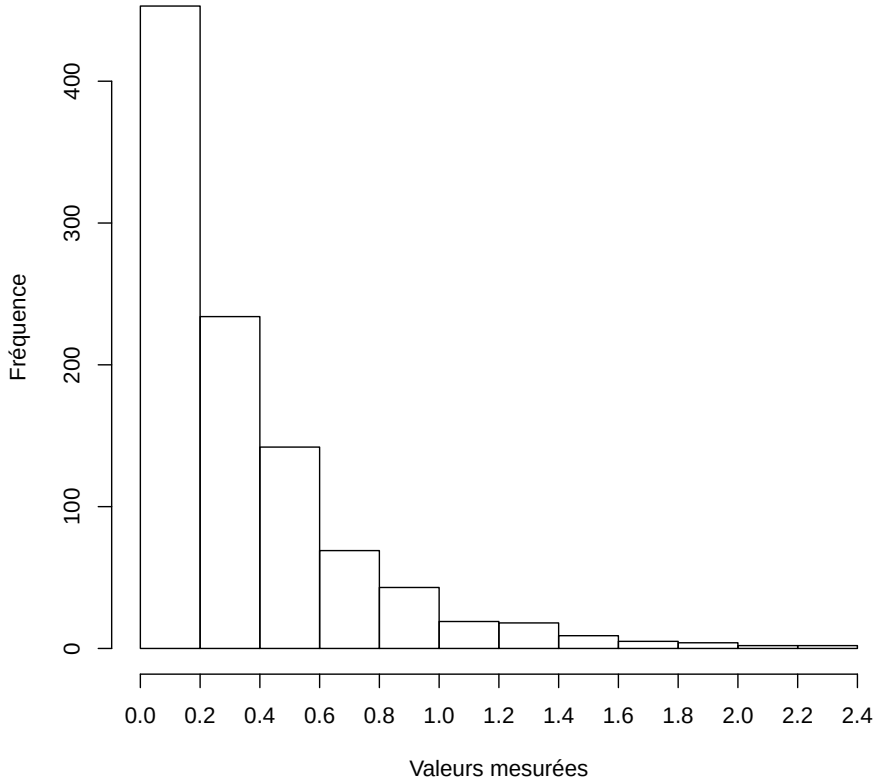


FIGURE 7.2 – Exemple d'histogramme illustré sous forme graphique sur base d'une liste de 1000 valeurs. Il peut par exemple être vu que plus de 400 valeurs mesurées se trouvaient dans l'intervalle $[0.0, 0.2)$.

Nous vous demandons d'écrire une fonction `histogramme()` qui prendra deux paramètres : une liste `L` de valeurs réelles (éventuellement négatives) ainsi qu'une liste `I` d'au moins deux valeurs réelles distinctes et triées dans l'ordre croissant. La liste `I` donne les bornes d'intervalles successifs sur la droite réelle. Par exemple, si cette liste est `[-5,2,9.5]`, elle symbolise deux intervalles : les valeurs entre -5 (inclus) et 2 (non inclus) ainsi qu'entre 2 (inclus) et 9.5 (non inclus).
Votre fonction `histogramme()` doit renvoyer un dictionnaire où chaque clef est un string de la forme `"[a,b)"` si l'on suppose que a est la borne inférieure (incluse) de l'intervalle et que b est la borne supérieure (non incluse) ; la valeur associée à chaque clef sera un entier qui dira combien de valeurs de la liste L sont incluses dans l'intervalle considéré.
Dans un second temps, nous vous demandons d'écrire une fonction `intervalles()` qui prendra trois paramètres : deux réels `inf` et `sup` (il est supposé que `sup` $>$ `inf`) ainsi qu'un entier strictement positif `n`. La fonction devra renvoyer une liste triée tel qu'utilisée par la fonction `histogramme()` et qui correspond à `n` intervalles de même taille telles que la borne inférieure du premier intervalle est `inf` et que la borne supérieure du dernier intervalle est `sup`.
Un exemple possible d'exécution (en mode interactif) serait le suivant :

```
>>> L = [-1, 0, 3, -10, 0.5, 9, 8, 12]
>>> I = intervalles(-7, 9.5, 4)
>>> histogramme(L, I)
{ '[-7,-2.875)': 0, '[-2.875,1.25)': 3,
 '[1.25,5.375)': 1, '[5.375,9.5)': 2 }
```

Vous ne pouvez pas utiliser la fonction `sort()` pour vous aider.

Chapitre 8

Tris

Choses à revoir

Revoir les tris suivants :

1. tri par énumération
2. tri bulle

Exercices préparatoires

Prép. 8.1. Triez à la main la liste suivante avec le tri par énumération, et puis avec le tri bulle :

$$V = \boxed{4}\boxed{8}\boxed{4}\boxed{2}\boxed{8}\boxed{1}$$

Exercices en séance

Ex. 8.1. Écrire une fonction triant un tableau par énumération (les valeurs possibles pour les composantes : $\{0, \ldots, 10\}$).
Pour rappel, le tri par énumération fonctionne comme suit, en supposant qu'on désire trier le tableau `V` :

1. On commence par *compter* le nombre d'occurrences de chaque valeur du tableau `V` et stocker ces valeurs dans un tableau `count`. À la fin de cette étape, `count`$[i]$ contiendra le nombre d'occurrences de la valeur i dans `V`.
2. On somme ensuite les valeurs contenues dans `count` de manière à ce que `count`$[i]$ contienne le nombre de valeurs $\leqslant i$ présentes dans `V`.
3. `count` permet alors de recopier les valeurs de `V` dans un nouveau tableau `W`, et ce, de manière triée. Pour ce faire, on parcourt `V` élément par élément. Chaque valeur v présente dans `V`, doit être placée dans la $(\texttt{count}[v] - 1)^{\text{e}}$ case de `W`, et `count`$[v]$ doit être décrémentée pour éviter les collisions.

Ex. 8.2. Implémentez le tri bulle. Cet algorithme fait avancer les plus grands éléments vers la fin du tableau au fur et à mesure de l'exécution de l'algorithme, à l'image de bulles dans un verre de Perrier remontant à la surface (la surface correspondant au côté droit du tableau).
À chaque itération, l'algorithme procède à des échanges successifs entre valeurs adjacentes : si une valeur en position i est plus grande que celle en $i + 1$, elles sont échangées. À la fin de la première itération de l'algorithme, nous avons dès lors le maximum du tableau dans la dernière

case. À chaque itération suivante, l'algorithme placera le maximum du reste du tableau dans la case juste avant le maximum précédent. Au bout de n itérations, le tableau sera dès lors bien trié.

Ex. 8.3. Implémentez le tri « *cocktail shaker* », variante du tri bulle. À chaque itération, l'algorithme fait d'abord en sorte qu'un maximum se retrouve le plus à droite possible du tableau. Deplus, il fait aussi en sorte qu'un minimum non encore traité se retrouve le plus à gauche possible. Une itération de l'algorithme place donc deux éléments à leur place : un maximum et un minimum.

Ex. 8.4. Implémentez le « *shotgun sort* ». Cet algorithme de tri se contente de mélanger aléatoirement tout un vecteur et de vérifier ensuite si le résultat est bien trié. Si ce n'est pas le cas, on recommence ces deux opérations. Pensez à utiliser la fonction `shuffle()` du module `random`. Quelles sont les complexités au meilleur et au pire cas de cet algorithme en sachant qu'un mélange se fait en $\mathcal{O}(n)$?

Ex. 8.5. Implémentez le tri « gnome » qui fonctionne comme suit en partant de l'élément en position 0 jusque $n - 2$ compris :

1. si $V[i] \leqslant V[i+1]$, incrémenter i ;
2. sinon, échanger $V[i]$ et $V[i+1]$ et décrémenter i si $i > 0$.

Ex. 8.6. Implémentez une étape de « *merge sort* ». Cet algorithme reçoit en paramètre deux listes triés : L_1 et L_2. Le résultat est une liste trié $result$ qui contient tous les éléments de L_1 et de L_2. Note : il ne suffit pas de concaténer les deux listes.

Ex. 8.7. ** Le tri « quicksort » consiste à placer un élément de la liste L (appelé pivot, qu'on prend par défaut comme le dernier élément de la liste) à sa place définitive, en permutant tous les éléments de telle sorte que tous ceux qui sont inférieurs au pivot soient à sa gauche et que tous ceux qui sont supérieurs au pivot soient à sa droite. Cette opération s'appelle le partitionnement. Pour chacune des sous-listes, on définit un nouveau pivot et on répète l'opération de partitionnement. Ce processus est répété récursivement, jusqu'à ce que l'ensemble des éléments soit trié.
Implémentez le tri « quicksort » décrit dans l'algorithme suivant. La fonction a la signature suivante : `quicksort(L, debut, fin)` où `L` est une liste d'éléments comparables, `debut` et `fin` sont des entiers correspondant à des indices dans la liste.

---

if $debut < fin$ **then**
  pivot = partitionner(L,debut,fin)
  quicksort(L,debut,pivot-1)
  quicksort(L,pivot+1,fin)
end if

Chapitre 9

Complexité

Exercices préparatoires

Révisez le chapitre complexité du cours.

Prép. 9.1. Trouvez la complexité au pire cas du code suivant :

```
i = 0
while i < n :
  t[i] = 0
  i += 3
```

Prép. 9.2. Trouvez la complexité au pire cas de la fonction suivante (`a`, `b` et `c` sont des vecteurs de taille `n*n`) :

```
def somme(c,a,b,n):
  for i in range(n):
    for j in range(n):
      c[i][j] = a[i][j] + b[j][i]
```

Prép. 9.3. Trouvez la complexité au pire cas du code suivant :

```
for j in range(4):
  for i in range(n):
    t[i] *= t[i]
```

Prép. 9.4. **(Examen de janvier 2010)** Donnez la complexité des opérations suivantes sur une liste Python en justifiant vos réponses.
– accéder à un élément d'un indice donné (`x = t[i]`)
– ajouter un élément en fin de liste (`t.append(3)`)
– insérer un élément (`t.insert(0,8)`)
– supprimer un élément (`t.remove(4)`)
– copier une liste
– concaténer 2 listes
– rechercher un élément dans une séquence (`x in s`)

Exercices en séance

Ex. 9.1. Trouvez la complexité au pire cas de la fonction suivante (`a`, `b` et `c` sont des vecteurs de taille `n*n`) :

```
def produit(c,a,b,n):
  for i in range(n):
    for j in range(n):
      c[i][j] = 0
      for k in range(n):
         c[i][j] += a[i][k] * b[k][j]
```

Ex. 9.2. Trouvez la complexité au pire cas du code suivant :

```
i = 0
j = 0

while j < n:
  t[j*n+i] = i
  if i == n-1:
    j += 1
    i = 0
  else:
    i += 1
```

Ex. 9.3. En faisant l'hypothèse que `n` est positif, donnez la complexité au pire cas du code suivant :

```
c = 0

i = n * n
while i > 0:
  j = n
  while j > 0:
    c += 1
    j = j//4
  i = i//2
print c
```

Ex. 9.4. Trouvez la complexité au pire cas de la fonction suivante (`a` est un vecteur de taille `n*n`) :

```
def symetrie(a,n):
  sym = True
  i = 0
  while i < n and sym:
    j = 0
    while j < i and sym:
      if a[i][j] != a [j][i]:
        sym = False
      j += 1
    i += 1
  return sym
```

Ex. 9.5. * Trouvez les complexités au pire cas et minimale du code suivant, en supposant que la fonction `boolrand` renvoie `True` ou `False` avec la même probabilité et qu'elle est de complexité constante.

```
for i in range(n):
  v[i] = True
for i in range(n-1):
  w[i] = boolrand()
w[n-1] = False
flag = True
j = 1
while flag:
  if v[j] == w[j] :
    for i in range(j,n-1) :
      w[i] = boolrand()
  else :
    flag = False
  j += 1
```

Ex. 9.6. * **(Examen d'août 2010)** On se donne deux fonctions, `expobeta` et `expo`, qui calculent toutes les deux la valeur du paramètre x élevée à la puissance n. On suppose x et n strictement positifs, et on ignore les problèmes de débordement.

1. Donnez les complexités aux pire et meilleur cas de ces deux fonctions sous la forme d'un grand $\mathcal{O}$ en fonction du paramètre n.
2. La fonction `expo` est-elle plus efficace ? Expliquez.

```
def expobeta(x,n):
  x1 = x
  for i in range(1,n):
    x *= x1
  return x

def expo(x,n):
  x1 = x
  i = 1
  while 2*i <= n:
    x = x*x
    i = 2*i
  while i < n:
    x *= x1
    i += 1
  return x
```

Ex. 9.7. * **(Examen d'août 2009)** Donnez la complexité au pire cas, en fonction de n et en utilisant la notation $\mathcal{O}$, des fonctions suivantes.

```
def f(V):
  n = len(V)
  i = 0
  j = 0
  while i < n:
    while j < i:
      V[j] *= i - j
      j += 1
    i += 3
```

```
def g(V):
  n = len(V)
  i = n - 1
  while i >= 0:
    V[i] = 0
    for j in range(i+1):
      V[i] += j
    i -= 1

def h(V):
  n = len(V)
  for i in range(n):
    V[i] = 0
    j = n
    while j > 1:
      V[i] += i*j
      j //= 2
```

Ex. 9.8. * **(Examen de juin 2010)**
Donnez la complexité de la fonction `fuse`, en tenant compte des différentes opérations et en justifiant vos réponses.

```
def fuse(t1, t2):
    if len(t1) == 0:
        return t2
    elif len(t2) == 0:
        return t1
    elif t1[0] < t2[0]:
        return [t1[0]] + fuse(t1[1:], t2)
    else:
        return [t2[0]] + fuse(t1, t2[1:])
```

Ex. 9.9. * **(Examen de janvier 2012)**

– Donnez la complexité du code suivant où n est un paramètre et mat une matrice de dimension $n \times n$ dont chaque élément contient un string de taille n :

```
sum = 0
for i in range(n):
   for j in range(n):
     if "a" in mat[i][j]:
       sum = sum + 1
```

– Donnez la complexité du code suivant où n est un paramètre :

```
i = 2
while i < n ** n:
   i = i * i
```

Ex. 9.10. * **(Examen de juin 2012)** Donnez la complexité au pire cas de l'algorithme suivant en fonction de `N`.

```
def incseq(V):
  N = len(V)
  i = 0
  k = 0
  maxseq = 0

  while i < N-1:
    seq = 1
    j = i
    done = False
    while j < N-1 and not done:
      if V[j] > V[j+1]:
        done = True
      else:
        seq += 1
      j += 1
    if seq > maxseq:
      maxseq = seq
      k = i
    i += seq

  return k
```

Chapitre 10

Logique et invariants

Exercices préparatoires

Révisez la matière concernée vue au cours ainsi que la logique de base telle que vue au cours de mathématiques. Nous vous invitons également à déjà essayer de réaliser l'exercice 10.1 du mieux que vous pouvez.

Rappels théoriques

Rappels de logique :

1. $\nexists i \cdot \phi(i) \iff \forall i \cdot \neg\phi(i)$ où ϕ est une formule logique fonction de i ;
2. $a \to b \to c \iff (a \wedge b) \to c$;
3. Priorité des opérateurs logiques : $\neg, \wedge, \vee, \to, \leftrightarrow$;
4. $\to$ est associatif à droite : $a \to b \to c$ est équivalent à $a \to (b \to c)$.

Formes à retenir :

1. $\forall i \cdot (0 \leqslant i < n) \to (\ldots)$
2. $\exists i \cdot (0 \leqslant i < n) \wedge (\ldots)$

Propriétés à respecter pour qu'un invariant Inv d'une boucle de condition C soit correct :

1. $Pre\ \{init\} \implies Inv$
2. $Inv \wedge C\ \{boucle\} \implies Inv$
3. $Inv \wedge \neg C\ \{fin\} \implies Post$

Preuve partielle : prouver que l'invariant d'une boucle est correct.

Preuve de terminaison : prouver qu'avec les préconditions et initialisations, la boucle se termine toujours.

Preuve totale : preuve partielle + preuve de terminaison.

Exercices en séance

Logique du premier ordre

Ex. 10.1. Exprimez en logique du premier ordre les affirmations suivantes (vous supposerez que tous les vecteurs sont de taille n) :

1. La variable `v` est positive ou nulle.
2. La variable `v` contient une valeur comprise entre 5 et 8 (bornes incluses).
3. Si la variable `v` est positive, alors la variable `w` est négative ou nulle.
4. Le vecteur `V` ne contient que des valeurs positives ou nulles.
5. Au moins une case du vecteur `V` contient une valeur supérieure ou égale à 5.
6. La variable `v` contient la somme des éléments du vecteur `W`.
7. La variable `v` contient la somme des `i` premiers éléments du vecteur `W`.
8. Les `i` premiers éléments du vecteur `V` sont inférieurs à 6.
9. Tous les éléments du vecteur `V` sont inférieurs ou égaux à `max`.
10. `max` est la valeur maximale contenue dans le vecteur.
11. Le vecteur `V` contient toutes des valeurs différentes.
12. Le vecteur `V` est trié (de façon croissante).
13. Les `i` premières cases du vecteur `V` sont triées (de façon decroissante).
14. Les cases comprises entre les indices k_1 et k_2 du vecteur `V` sont triées (de façon non-decroissante).
15. Si la variable `i` est positive, les `i` premières cases du vecteur `V` sont triées de façon non-decroissante ; sinon, le vecteur `V` est initialisé à 0.
16. La partie du vecteur `V` comprise entre les indices k_1 et k_2 est triée, et il n'existe pas de plus grande partie (en terme de nombre de cases) du vecteur qui soit triée.

Ex. 10.2.* (Examen d'août 2010) Exprimez en logique du premier ordre que le vecteur V à n composantes entières d'indices $\{0, 1, \ldots, n-1\}$:

1. est tel que toutes ses composantes (exceptée celle d'indice 0) ont une valeur qui est un multiple de la composante précédente.
2. est tel que chacune de ses composantes a une valeur qui est un multiple de son indice correspondant.

Ex. 10.3.* (Examen de janvier 2010) Pour le code suivant, sachant que les valeurs lues dans la matrice `M` de dimensions `MAX` $\times$ `MAX` sont toutes dans l'intervalle $[0, \texttt{MAX} \times \texttt{MAX} - 1]$, exprimez en logique du premier ordre la condition sur le contenu de `M` pour que le code renvoie un résultat vrai. Donnez aussi la complexité au pire cas de l'algorithme. Justifiez vos réponses.

```
def condition(M):
  MAX = len(M)
  V = [False] * (MAX*MAX)
  for i in range(MAX):
    for j in range(MAX):
      V[M[i][j]] = True
  n = 0
  while n < MAX*MAX and V[n]:
    n += 1
  return n == MAX*MAX
```

Ex. 10.4.* (Examen de janvier 2010) Pour le code suivant, sachant que les valeurs lues dans la matrice `M` de dimensions `MAX` $\times$ `MAX` sont toutes dans l'intervalle $[0, \texttt{MAX} \times \texttt{MAX} - 1]$, exprimez en logique du premier ordre la condition sur le contenu de `M` pour que le code renvoie un résultat vrai. Donnez aussi la complexité au pire cas de l'algorithme. Justifiez vos réponses.

```
def condition(M):
  MAX = len(M)
  found = True
  k = 0
  while k < MAX*MAX and found:
    found = False
    i = 0
    while i < MAX and not found:
      j = 0
      while j < MAX and not found:
        found = (M[i][j] == k)
        j += 1
      i += 1
    k += 1
  return found
```

Ex. 10.5. * **(Examen de juin 2010)** Soit une liste d'entiers V à n éléments (n strictement positif). Exprimez en logique du premier ordre que

1. V ne contient pas deux éléments de valeur égale ;
2. V est tel que ses éléments de **valeur** paire sont triés par ordre croissant et ses éléments d'**indice** impair sont triés par ordre décroissant.

Ex. 10.6. * **(Examen de juin 2010)** Pour le code suivant, où n est une constante entière strictement positive et en ayant des valeurs entières quelconques dans la matrice M (carrée de côté n), exprimez en logique du premier ordre la condition sur le contenu de M pour que le code renvoie un résultat vrai. Donnez également la complexité au pire cas de l'algorithme.

```
def condition(M):
  n = len(M)
  found = True
  i = 0
  while found and i < n:
    found = False
    j = 0
    while not found and j < n-1:
      k = j+1
      while not found and k < n:
        found = (M[i][j] == M[i][k])
        k += 1
      j += 1
    i += 1
  return found
```

Ex. 10.7. * **(Examen d'août 2009)** La fonction `test` ci-dessous reçoit un vecteur en paramètre et vérifie si celui-ci satisfait une certaine propriété. Donnez, d'abord en français puis sous forme d'une formule logique, la propriété testée. En d'autres termes, exprimez une condition nécessaire et suffisante sur le vecteur pour que la fonction `test` renvoie la valeur `True`.

```
def test(V):
  n = len(V)
  stop = False
  if n < 3:
    return True
  diff = V[1] - V[0]
```

```
  i = 1

  while not stop and i < n-1:
    nvdiff = V[i+1] - V[i]
    stop = (diff > nvdiff)
    diff = nvdiff
    i += 1

  return not stop
```

Ex. 10.8. * **(Examen de juin 2011)** L'algorithme suivant fusionne 2 listes triées.

1. Exprimez en logique des prédicats, les préconditions (hypothèses sur les paramètres) et la postcondition (résultats) de la fonction `fuse`.
2. Donnez la complexité (au pire cas) de la fonction `fuse`, en tenant compte des différentes opérations et en justifiant vos réponses (sans justification, la réponse est nulle).

```
def fuse(t1, t2):
  if len(t1) == 0:
    return t2
  elif len(t2) == 0:
    return t1
  elif t1[0] < t2[0]:
    return [t1[0]] + fuse(t1[1:], t2)
  else:
    return [t2[0]] + fuse(t1, t2[1:])
```

Invariants

Ex. 10.9. Vérifiez que $(0 \leqslant \texttt{i} \leqslant \texttt{n}) \wedge (\texttt{j} = \texttt{i} \times \texttt{m})$ est un invariant correct pour la boucle suivante (on suppose que `m` et `n` sont positifs) :

```
i = 0
j = 0
while i != n:
  j += m
  i += 1
```

Ex. 10.10. Soit le morceau de code suivant (on suppose que n est strictement positif et que V et W sont des vecteurs d'entiers de dimension n) :

```
i = 0
s = 0
while i < n:
  s += V[i]*W[i]
  i += 1
```

1. Expliquez en français ce que fait cette boucle.
2. Exprimez à l'aide d'une formule logique les précondition et postcondition de la boucle.
3. Démontrez que

$$(0 \leqslant i \leqslant n) \wedge \left(s = \sum_{j=0}^{i-1} (V[j] \times W[j]) \right)$$

est un invariant correct pour la boucle.

4. Servez-vous de ces informations pour démontrer que la boucle fait bien ce que vous aviez prévu (en supposant qu'elle se termine).

Ex. 10.11. À quoi sert la boucle suivante ?

```
q = 0
r = j1
while r >= j2:
  q += 1
  r -= j2
```

Démontrez que cette boucle est correcte (qu'elle fait bien ce que vous venez de prédire) en supposant que `j1` soit un entier positif et `j2` un entier strictement positif. Pour ce faire, vérifiez que $Inv \equiv (\mathtt{r} + \mathtt{q} \times \mathtt{j2} = \mathtt{j1}) \wedge (q \in \mathbb{N}) \wedge (r \in \mathbb{N})$ est un invariant, et donnez la postcondition du `while`.

Ex. 10.12. Voici la fonction $plus$:

```
def plus (x,y):
  while y>0:
    x += 1
    y -= 1
  return x
```

Démontrez que cette boucle calcule bien la somme de `x` et `y` dans `x`. Pour ce faire, montrez que l'invariant du `while` est $Inv \equiv (\mathtt{y} \geqslant 0) \wedge (\mathtt{x} + \mathtt{y} = x_0 + y_0)$, où x_0 et y_0 dénotent les valeurs initiales de x et y respectivement, et montrez la terminaison de la fonction. On suppose initialement que $y \geqslant 0$.

Ex. 10.13. La boucle suivante est censée imprimer la valeur maximale de `v`. Démontrez que $Inv \equiv \forall j \cdot (0 \leqslant j \leqslant i < n) \rightarrow (V[i] \geqslant V[j])$ est un invariant correct pour la boucle.

```
i = 0
while i < n-1 :
  if v[i] > v[i+1] :
    v[i],v[i+1] = v[i+1],v[i]
  i += 1
print(v[n-1])
```

Ex. 10.14. * (**Question de janvier 2003**) On se donne la fonction suivante, qui fait l'hypothèse que le vecteur `V` passé en paramètre est trié.

```
def f(V,x):
  n = len(V)
  bi = 0
  bs = n - 1
  m = (n - 1) // 2

  while (bs >= bi) and (V[m] != x) :
    if V[m] < x :
      bi = m + 1
    else :
      bs = m - 1
    m = (bi + bs) // 2

  if x != V[m] :
    m = -1

  return m
```

1. Quelle est l'utilité de cette fonction ?
2. Montrez que

$$\Big(\big(\exists i \cdot 0 \leqslant i < n \wedge V[i] = x\big) \rightarrow bi \leqslant i \leqslant bs\Big)$$
$$\wedge (0 \leqslant bi < n + 1) \wedge (-1 \leqslant bs < n) \wedge m = \left\lfloor \tfrac{bi+bs}{2} \right\rfloor$$

   est un invariant correct pour la boucle.
3. Démontrez la terminaison de la boucle.

Ex. 10.15.* (Examen de janvier 2005) Soit le code suivant qui suppose que `n` est une constante entière strictement positive.

```
def test(V):
  n = len(V)
  delta = 0
  i = 0
  while i < n-1 and V[i] + delta <= V[i+1]:
    delta = V[i+1] - V[i]
    i += 1
  return i == n-1
```

1. Exprimez en français et en logique du premier ordre la condition nécessaire et suffisante sur le vecteur V pour que la fonction `test` renvoie la valeur *True*.
2. Montrez que

$$\forall j \cdot (0 \leqslant j < i - 1) \rightarrow \big(V[j+1] - V[j] \leqslant V[j+2] - V[j+1]\big)$$

   est un invariant correct pour la boucle.
3. Démontrez la terminaison de la boucle.

Chapitre 11

Récursivité

Exercices préparatoires

Matière à réviser :
– la gestion de mémoire (*stack frames*) ;
– les fonctions.

Prép. 11.1. Une fonction `S(n)` qui calcule la somme des `n` premiers entiers (i.e. $S_n = \sum_{i=1}^{n} i$).

Prép. 11.2. Une fonction `F(n)` qui calcule la n-ème valeur de la suite F_n où $F_0 = 0$, $F_1 = 1$ et $F_n = F_{n-2} + F_{n-1}$.

Prép. 11.3. Expliquez les avantages et inconvénients des procédures récursives (c'est-à-dire où on empile des appels récursifs de fonctions) vis-à-vis des procédures itératives (boucles).

Exercices en séance

Écrivez les fonctions suivantes de manière récursive :

Ex. 11.1. Une fonction `ackermann(m,n)` qui implémente la fonction d'Ackermann $A(m, n)$ définie comme suit :

$$A(m,n) = \begin{cases} n+1 & \text{si } m = 0 \\ A(m-1,1) & \text{si } m > 0 \text{ et } n = 0 \\ A(m-1, A(m,n-1)) & \text{si } m > 0 \text{ et } n > 0. \end{cases}$$

Vérifiez que `ackermann(3,6)` donne 509. Que se passe-t-il pour de plus grandes valeurs ? Pourquoi ?

Ex. 11.2. Une fonction `pgcd(x,y)` qui calcule le plus grand commun diviseur de deux entiers positifs `x` et `y`. Pour cela, utilisez la méthode d'Euclide. Cette méthode se base sur l'observation que si r est le reste de la division de x par y, alors le pgcd de x et y est égal au pgcd de y et r. Considérez comme cas de base que le pgcd de x et 0 est x.

Ex. 11.3. Une fonction `factorielle(n)` qui calcule et renvoie la factorielle de `n` (noté $n!$). Pour rappel : $n! = 1 \times 2 \times 3 \ldots \times n$.

Ex. 11.4. Une fonction `puissance(x,n)` qui calcule et renvoie la n-ième ($n \in \mathbb{N}$) puissance de `x` sans utiliser l'opérateur `**`.

Ex. 11.5. Une fonction `triangle_pascal(i,j)` qui, étant donné deux entiers positifs `i` et `j`, renvoie la valeur située à la i^{e} ligne et la j^{e} colonne du triangle de Pascal. Le triangle de Pascal (voir Figure 11.1) est construit comme suit :
– La valeur en $(0, 0)$ vaut 1.
– Pour toute case du triangle, la valeur de cette case est la somme de la valeur située au-dessus et de celle située au-dessus à gauche.
– Pour toutes les autres « cases », considérez une valeur nulle.

	0	1	2	3	4	5
0	1					
1	1	1				
2	1	2	1			
3	1	3	3	1		
4	1	4	6	4	1	
5	...					

FIGURE 11.1 – Triangle de Pascal

Ex. 11.6. Une fonction `contient(n,d)` qui renvoie `True` si et seulement si l'entier positif représenté par `n` contient le chiffre $d \in \{0, \ldots, 9\}$ représenté par `d` (astuce : utilisez la division entière et le modulo).

Ex. 11.7. Une fonction `inverse(n)` qui renvoie le miroir d'un entier positif n. Par exemple le miroir de 475 est 574.

Ex. 11.8. Une fonction `est_multiple(n,d)` qui renvoie `True` si et seulement si l'entier positif `n` est un multiple de l'entier positif `d`. Vous ne pouvez pas utiliser les opérateurs `//`, ni `/`, ni `%`.

Ex. 11.9. Une fonction `divE(n,d)` qui renvoie la division entière entre n et d (i.e. $n//d$) sans utiliser l'opérateur $//$.

Ex. 11.10. * Une fonction `concatenation(L1,L2)` qui renvoie une liste `L` - la concatenation de listes `L1` et `L2`. par exemple : si `L1 = [1,2]` et `L2 = [3,4]`, alors la fonction `concatenation(L1,L2)` doit renvoyer la liste `[1,2,3,4]`. Vous ne pouvez pas utiliser l'opérateur +, ni la méthode `extend()`.

Chapitre 12

Fichiers et exceptions

Exercices préparatoires

Matière à réviser :
– la fonction `open(file,mode)` avec modes d'ouverture *read*, *write* et *append* ;
– les méthodes `close()`, `write()`, `read()`, `readline()`, `readlines()`, `seek()` sur les objets fichiers ;
– le module `os` et la notion de chemins *absolus* et *relatifs* ;
– les exception plus fréquentes : `ValueError`, `TypeError`, `IndexError`, `IOError` ;
– lever une exception via `raise` ;
– gérer des exceptions via `try-except` ;
– les fonctions `ord()` et `chr()` (voir exercice 4.8, p. 19).
Lisez le *Mode d'emploi introductif pour les salles du NO4 et NO3*, partie « la console Linux ».

Prép. 12.1. Écrivez une fonction `lecture(fichier)` qui affiche le contenu du `fichier`.

Prép. 12.2. Écrivez une fonction `ecriture(fichier,chaine)` qui crée un fichier vide portant le nom `fichier` (ou, s'il existe, vide ce fichier) et écrit la chaîne de caractères `chaine` dans celui-ci.

Prép. 12.3. Écrivez une fonction `ajouter(fichier,chaine)` qui ajoute la chaîne de caractères `chaine` à la fin du fichier dont le nom est la chaîne de caractères `fichier`.

Prép. 12.4. Les fonctions manipulant des fichiers lèvent, en cas de problème, une exception de type `IOError`. Écrivez un code principal utilisant la fonction écrite à l'exercice préparatoire 12.1 et qui gère correctement cette exception. Ainsi, si on tente d'ouvrir un fichier inexistant en lecture, votre code devra le signaler par un message d'erreur à l'utilisateur plutôt que de faire planter l'interpréteur.

Prép. 12.5. Montrez quels seront les résultats du code ci-dessous. Expliquez brièvement ce que doit contenir le fichier lu pour déclencher chacun des cas d'exception.

```
try:
  f = open('myfile.txt')
  s = f.readline()
  i = int(s.strip())
except IOError:
  print("I/O error.")
except ValueError:
  print("Could not convert data to an integer.")
```

```
except:
  print("Unexpected error.")
  raise
```

Exercices en séance

Ex. 12.1. Écrivez une fonction `wc(fichier)` qui doit ouvrir le fichier en question et renvoyer le nombre de caractères (sans compter les caractères de retour à la ligne), le nombre de mots et le nombre des lignes du fichier. Nous définissons ici un mot comme étant une chaîne de caractères (maximale) répondant *True* à la méthode `isalnum()`.

Ex. 12.2. Écrivez une fonction `replace(in_path, out_path, pattern, subst)` qui doit ouvrir le fichier dont le chemin est `in_path` et remplacer dans ce dernier toutes les occurrences du string `pattern` par `subst`. Le résultat de cette modification devra être écrite dans le fichier de chemin `out_path`.

Ex. 12.3. À l'aide d'un dictionnaire, écrivez les fonctions suivantes :
- `file_histogram(file)` qui prend en paramètre le nom, sous forme d'une chaîne de caractères, d'un fichier texte `file` et qui renvoie un dictionnaire contenant la fréquence absolue (c'est-à-dire le nombre d'occurrences) de chaque lettre dans le texte contenu dans le fichier.
- `vowels_histogram(file)` qui prend en paramètre le nom, sous forme d'une chaîne de caractères, d'un fichier texte `file` et qui renvoie un dictionnaire contenant la fréquence absolue des chaque suite de voyelles (par exemple : « i », « e », « oe », « oui », « eui », « you », etc.). Notez que si on trouve « oe » par exemple, il ne faut pas compter une occurrence de « o » ni de « e ».
- `words_by_length(file)` qui prend en paramètre le nom, sous forme d'une chaîne de caractères, d'un fichier texte `file` et qui renvoie un dictionnaire associant à une valeur entière ℓ, la liste des mots dans le fichier de longueur ℓ.

Ex. 12.4. Reprenez la fonction `trace(M)` écrite à l'exercice 6.1 (p. 26) et ajoutez-y la capture avec une ou plusieurs exceptions des erreurs pouvant survenir dans le cas où la matrice ne serait pas adaptée pour le calcul de trace (c'est-à-dire quand elle n'est pas carrée ou que les éléments de la diagonale ne peuvent être sommés, par exemple si on a un mélange de nombres et de chaînes de caractères).

```
def trace(M):
  dimension = len(M)
  if dimension != len(M[0]):
    res = None
  else:
    res = 0
    for i in range(dimension):
      res += M[i][i]
  return res
```

Ex. 12.5. **Pendu (mini-projet)** Nous vous demandons de développer un jeu du pendu. Dans ce dernier, le joueur essaie de deviner en un nombre limité de tentatives, lettre par lettre, un mot choisi par l'ordinateur. Ce dernier choisira au hasard un des mots contenu dans le fichier `dictionary.txt`. Le joueur n'a droit qu'à dix tentatives infructueuses (à la 10^{e} erreur, le joueur a perdu). À chaque étape, un récapitulatif de l'état du jeu (c'est-à-dire le nombre restant

de tentatives infructueuses, les lettres déjà jouées, ainsi que le mot à deviner dans lequel chaque lettre encore non trouvée est représentée par une astérisque) doit être affiché avant le choix suivant de lettre du joueur.
À chaque fois que le joueur entrera une lettre, le programme vérifiera si ce dernier est dans le mot caché. Si c'est le cas, il ajoute les nouvelles lettres dévoilées, sinon le nombre de tentatives restantes est décrémenté. Si le nombre de tentatives restantes atteint 0, le joueur a perdu. Si l'entièreté du mot est dévoilé, la partie est gagnée et le score est affiché. Ce dernier est calculé à l'aide de la formule suivante :

$$\text{score} = \ell + r$$

où ℓ est la longueur du mot et r le nombre de tentatives restantes. Pour le choix au hasard du mot, nous vous demandons d'utiliser la bibliothèque `random` et `randint()` plus particulièrement.

Ex. 12.6. Checksum (mini-projet) Les donnes peuvent parfois être perdu ou corrompu pendant leur transmission. On vous demande d'implémenter un *code détecteur d'erreur* et un *code correcteur d'erreur*. Ces codes permettent de détecter de de corriger les erreurs éventuelles de transmission.
Dans ce mini-projet supposez que le fichier texte ne contient que des caractères 0 et des caractères 1 et que tous les lignes ont la même longueur.

Code détecteur L'idée de ce code est d'ajouter des informations redondantes dans le message. Nous allons diviser le fichier en bloques et ajouter un bit à chaque bloque pour que le nombre de bits à 1 soit impaire. Tous les nouveaux bits (les bits de parité) qu'on calcule ainsi constituent un *checksum*.

1. Écrivez une fonction $addChecksum(filename)$ qui lit le contenu d'un fichier, calcule le checksum et l'ajoute à la fin de fichier. Prenez donc une ligne pour un bloque, voir exemple 12.1.
2. Écrivez une fonction $verifyChecksum(filename)$ qui lit un fichier avec le checksum et vérifie si le checksum est correcte (renvoie $True$ si c'est le cas).

File :

```
1 0 1 1
0 0 0 1
1 0 1 0
```

Les bits de parité qu'il faut ajouter : 0 (ligne 1), 0 (ligne 2), 1 (ligne 3).
Apres l'execution de addChecksum :

```
1 0 1 1
0 0 0 1
1 0 1 0
0 0 1
```

FIGURE 12.1 – Exemple addChecksum(filename).

Code correcteur L'idée de ce code est de modifier le code détecteur, pour pouvoir aussi corriger des erreurs. On va donc ajouter les bits de parité pour chaque ligne et pour chaque colonne, voir l'exemple Figure 12.2 . Ce code permet de corriger au plus une erreur.

1. Ecrivez une fonction $addChecksumCorr(filename)$ qui lit le contenu d'un fichier, calcule le checksum et l'ajoute à la fin de fichier.
2. Ecrivez une fonction $verifyAndCorrect(filename)$ qui lit un fichier avec le checksum, verifie si le checksum est correcte et corrige le fichier si le fichier est corrompu.

File :

```
1 0 0 1
0 0 0 1
1 0 1 0
```

Calcule de bits de parité :

```
1 0 0 1  1
0 0 0 1  0
1 0 1 0  1

1 1 0 1
```

Fichier apres l'execution de addChecksumCorr :

```
1 0 1 1
0 0 0 1
1 0 1 0
1 0 1
1 1 0 1
```

FIGURE 12.2 – Exemple addChecksum(filename).

Ex. 12.7. ** Faites l'exercice 12.6 sans supposer que le fichier contient que des 1 et des 0 et sans supposer que tous les lignes ont la même longueur.

Chapitre 13

Introduction aux classes

Exercices préparatoires

Matière à réviser :
– notions de classe, instance, attribut,
– différence entre fonction pure et méthode d'instance,
– méthode : `__init__`, `__str__`, `__repr__`,
– surcharge d'opérateur.

Prép. 13.1. Que fait le code suivant ? Expliquez.

```
class A:
  def __init__(self,msg="bright"):
    self.attr = msg

  def __repr__(self):
    return self.attr

  def set(self, msg):
    self.attr = msg

x = A()
print(x)
y = A("side of life")
print(y)
y.set(x.attr)
print(y)
```

Prép. 13.2. Écrivez une classe `Point` qui va définir un point dans un espace de dimension 3. Nous vous demandons de définir un constructeur avec valeurs par défaut, de définir la méthode `__repr__(self)` de la classe pour pouvoir utiliser `print()` sur vos objets Point et de définir la méthode `distance(self,point)` qui prend un objet de type Point en entrée et qui renvoie la distance entre les deux points.

Exercices en séance

Ex. 13.1. Écrivez une classe `StopWatch` qui modélise un chronomètre. Cette classe doit contenir les méthodes suivantes :
– `start` qui lance le chronomètre.

– `stop` qui arrête le chronomètre.
– `getElapsedTime` qui renvoie le temps écoulé en secondes depuis le dernier lancement du chronomètre (s'il est en train de tourner), ou le temps écoulé entre le dernier départ et le dernier arrêt si le chronomètre a été arrêté. Si le chronomètre n'a jamais été lancé, cette méthode devra renvoyer 0.

Conseil : utilisez le module `time` fourni par Python, en particulier la fonction `time.time()`.

Ex. 13.2. Écrivez une classe `Monome`. Un monôme est un polynôme composé d'un seul terme. Par exemple, $3x^4$ ou $7x^5$ ou encore 8 sont des monômes. Un monôme est caractérisé par deux attributs : son coefficient et son degré. Votre classe devra donc comporter deux attributs, une méthode d'initialisation avec valeurs par défaut ainsi que deux méthodes d'accès (*getters*) qui renvoient les valeurs des attributs. Enfin, la méthode `__repr__` est demandée.

Ex. 13.3. Définissez les méthodes `__add__`, `__sub__`, `__mul__`, `__floatdiv__` ,`__truediv__`, `__lt__`, `__gt__`, `__eq__` pour la classe `Monome` définie précédemment en respectant les contraintes suivantes :
– le méthode `__add__`, `__sub__`, `__mul__`, `__truediv__`, `__floatdiv__`, doivent renvoyer un monôme ;
– si les monômes ont des degrés différents, les méthodes `__add__` et `__sub__`, doivent lever une exception ;
– nous souhaitons que la relation d'ordre entre deux monômes soit la suivante : $a^x \ \langle op \rangle \ b^y \iff x \ \langle op \rangle \ y$, où $\langle op \rangle$ est un opérateur de l'ensemble $\{<, \leq, =, \geq, >\}$.

Exemple : $2x^5 > 3x^4$ est vrai .

Ex. 13.4. Écrivez une classe `Polynome` qui va définir un polynôme. Un polynôme est une somme finie de monômes ; nous pouvons donc représenter un polynôme par une liste de monômes. Nous vous demandons d'écrire les méthodes `__init__`, `__repr__`. La méthode `__repr__(self)` devra formater l'affichage du polynôme en respectant l'ordre des monômes tel que donné à l'exercice 13.3 :

```
>>> p = Polynome([Monome(-2,4), Monome(3,5), Monome(9,0), Monome(-11,1)])
>>> print(p)
Polynome : 3 x^5 - 2 x^4 - 11 x + 9
```

Ex. 13.5. Ajoutez à la classe `Polynome` la méthode `reduce` qui va faire l'addition entre monômes du même degré.

Exemple : $3x^5+2x^4+3x^4+2x^2+11x+3x+9+10$ devient $3x^5+5x^4+2x^2+14x+19$.

Ex. 13.6. * Définissez les méthodes `__add__` et `__mult__` sur la classe `Polynome` qui permettent de faire l'addition et la multiplication entre deux polynômes.

```
>>> p1 = Polynome([Monome(2,4), Monome(3,5), Monome(9,0)])
>>> p2 = Polynome([Monome(2,5), Monome(4,0), Monome(10,1)])
>>> p3 = p1 + p2
>>> p4 = p1 * p2
>>> print(p3)
Polynome : 5 x^5 + 2 x^4 + 10 x + 13
>>> print(p4)
Polynome : 6 x^10 + 4 x^9 + 30 x^6 + 50 x^5 + 8 x^4 + 90 x + 36
```

Ex. 13.7. Écrivez une classe `Rationnel` qui va définir un nombre rationnel. Un nombre rationnel est un nombre qui peut s'exprimer comme le quotient de deux entiers $\frac{a}{b}$, où a et b sont deux entiers (avec b non nul). On appelle a le numérateur et b le dénominateur. Votre classe devra avoir deux champs attributs et définir les méthodes suivantes : (`__init__`, `__add__`, `__sub__`, `__mul__`, `__floatdiv__`, `__truediv__`, `__str__`, `__float__`).

Ex. 13.8. ** Ajoutez à la classe `Polynome` la méthode `composition` qui va faire la composition entre deux polynômes.

Exemple : Si pour tout réel x :

$$\begin{aligned} f(x) &= -3.x^2 + 4.x - 2 \\ g(x) &= 4.x - 1 \end{aligned}$$

alors le polynôme $f \circ g$ est le polynôme défini pour tout réel x par :

$$\begin{aligned} (f \circ g)(x) &= f\big(g(x)\big) \\ &= -3.[\ g(x)\ ]^2 + 4.[\ g(x)\ ] - 2 \\ &= -3.[\ 4.x - 1\ ]^2 + 4.[\ 4.x - 1\ ] - 2 \\ &= -3.[\ 16.x^2 - 8.x + 1\ ] + 4.[\ 4.x - 1\ ] - 2 \\ &= -48.x^2 + 24.x - 3 + 16.x - 4 - 2 \\ &= -48.x^2 + 40.x - 9 \end{aligned}$$

Par contre, $g \circ f$ est le polynôme défini pour tout réel x par :

$$\begin{aligned} (g \circ f)(x) &= g\big(f(x)\big) \\ &= 4.[\ f(x)\ ] - 1 \\ &= 4.[\ -3.x^2 + 4.x - 2\ ] - 1 \\ &= -12.x^2 + 16.x - 8 - 1 \\ &= -12.x^2 + 16.x - 9 \end{aligned}$$

Chapitre 14

Base de données

Exercices préparatoires

Prép. 14.1. Les bases de données sont utilisées pour stocker un grand nombre de données de manière structurée et optimisée afin de pouvoir accéder rapidement et facilement à une information. Dans une base de données, nous retrouvons plusieurs tables (que nous pouvons considérer comme différents ensembles contenant des informations différentes : des contacts, des véhicules, des adresses, etc.). Chacune contient plusieurs lignes (entrées) ; chacune d'entre elles correspond à une entité (une personne, un événement ou un véhicule par exemple).
Dans une table, une entrée est définie par une clé. Celle-ci représente l'entrée de manière unique dans la table : une clé ne peut pas correspondre à deux ou plus d'entrées dans une table. Nous allons représenter une table d'une base de données à l'aide d'un dictionnaire et de tuples.
Pour cet exercice, nous allons considérer une table d'étudiants contenant leur nom, âge et taille, comme par exemple à la Table 14.1. Le nom de l'étudiant servira de clé dans la table. Nous ne pourrons donc pas avoir deux entrées dans la table pour un même étudiant.

clé — Etudiants

nom	*age*	*taille*
. . .	. . .	. . .
Arnaud	35	1.78
. . .	. . .	. . .

TABLE 14.1 – Extrait d'une table pour stocker des étudiants dans une base de données.

Nous vous demandons d'écrire la fonction de remplissage `ajouter(table)` qui doit demander les données à l'utilisateur jusqu'à ce que ce dernier entre `none` au clavier et rajouter les informations dans la table `table` donnée en paramètre.
Vous devez ensuite écrire la fonction de consultation `consulter(table)` qui doit elle aussi demander à l'utilisateur de rentrer le nom d'un étudiant présent dans la table `table` donnée

en paramètre et dont on veut consulter les informations (et ce jusqu'à ce que l'utilisateur rentre `none` au clavier). La fonction doit afficher ses informations de la manière suivante :

```
"Nom : Jeannot Lapin - age : 23 ans - taille : 1.30m"
```

Exercices en séance

Nous allons pour cette Super Séance perfectionner notre base de données commencée à l'exercice 14.1.

Ex. 14.1. Reprenez la fonction `ajouter(table)` écrite pour l'Ex. 14.1 sur les dictionnaires et écrivez comment capturer une exception dans le cas où un les paramètres passés par l'utilisateur ne sont pas du bon type. Dans le cas où l'exception est levée, vous afficherez un message d'erreur décrivant.

```
def ajouter(table):
  rep = input("Nom ? ")
  while rep != None :
    if rep not in table:
      age = input("Âge ? ")
      taille = input("Taille ? ")
      table[rep] = (age,taille)
    else:
      print ("Erreur : l'étudiant est déjà encodé.")
    rep = raw_input("Nom ? (none pour arrêter) ")
  return table
```

Ex. 14.2. Écrivez une fonction `conversion(table,critere)` dont l'objectif est de changer le critère utilisé comme clé dans une table. Comme à l'Ex. 14.1, chaque entrée de la table fournie à la fonction est composée d'une clé et d'un tuple comprenant les valeurs des autres critères. La variable `critere` est l'indice, dans le tuple, de la clé du nouveau dictionnaire. Par exemple, si nous reprenons l'Ex. 14.1, l'objectif de la fonction, si on lui passe `0` comme critère, serait de construire un nouveau dictionnaire tel que la clef est l'âge de l'étudiant, tandis que le nom (qui était la clef du dictionnaire d'origine) remplacerait la valeur de la colonne « âge ».
Si le nouveau critère n'est pas unique pour les différentes entrées de la table, les nouveaux tuples seront insérés dans des listes. Vous considérerez que le nouveau critère est toujours utilisable comme clé (on peut donc supposer que ce n'est jamais une liste ou un dictionnaire par exemple).
Voici un exemple de conversion :

```
>>> t = {'Kate' : (35, 1.90), 'George' : (35, 1.99), 'Jamie' : (23, 1.73)}

>>> print (convert(t,0))
{35: [('Kate', 1.90), ('George', 1.99)], 23: [('Jamie', 1.73)]}

>>> print (convert(t,1))
{1.90: [(35, 'Kate')], 1.73: [(23, 'Jamie')], 1.99: [(35, 'George')]}
```

Ex. 14.3. L'exercice 14.1 n'est pas très pratique car, après avoir exécuté le code et fermé l'interpréteur, le contenu de la table n'existe plus au prochain lancement de l'interpréteur. Pour maintenir les données persistantes d'une exécution à l'autre, nous allons donc les sauvegarder dans un fichier.

Nous vous demandons d'écrire deux fonctions : `sauvegarder(table,fichier)` qui sauvegarde une table dans un fichier et `charger(fichier)` qui renvoie une table remplie à partir d'un fichier (nous gardons la même conception de table que pour l'exercice 14.1).
Chaque ligne du fichier texte correspondra à un élément du dictionnaire (de la table) que l'on désire sauvegarder. Elle sera formatée de manière à bien séparer les informations. Nous respecterons le format suivant :

```
nom_de_letudiant@age#taille
```

Voici un exemple :

```
Jerome@24#1.74
Naim@25#1.76
...
```

Pour ouvrir un fichier en écriture, il faut ajouter un paramètre supplémentaire à la fonction `open`. Il nous faut donc utiliser l'instruction :

```
fichier = open('nom_du_fichier','w')
```

où `'w'` permet d'indiquer que nous ouvrons le fichier en écriture *(write)*.

Ex. 14.4. Stocker de l'information est une chose pratique, mais pouvoir la modifier en est une autre chose tout aussi importante et utile.
Une table contient plusieurs entrées, il faut donc toujours préciser quelle ligne on désire modifier. Nous supposons ici que la clé d'une entrée ne peut jamais être modifiée.
La fonction `update(table)` reçoit en paramètre une table contenant des étudiants et qui demandera le nom de l'étudiant dont on veut modifier les informations. Si l'étudiant n'existe pas, la fonction lèvera une exception (via `raise` par exemple) de type `KeyError` qui servira à afficher un message d'erreur plus tard.

La fonction devra alors parcourir les champs de l'entrée indiquée en demandant si l'on désire en modifier la valeur :

```
> Veuillez fournir le nom de l'étudiant à modifier :
> Jerome
> L'age de Jerome est : 24 (entrez une nouvelle valeur ou \
 entrez 0 pour ne pas modifier cette valeur)
> 0
> La taille de Jerome est : 1.74 (entrez une nouvelle valeur ou \
 entrez 0 pour ne pas modifier cette valeur)
> 1.77
```

Ex. 14.5. Il peut être aussi utile de pouvoir supprimer une entrée d'une table. À nouveau, il nous faut alors préciser la table et l'entrée que nous désirons y retirer.
Écrivez une fonction `delete(table)` qui demande le nom d'un l'étudiant et qui retire l'entrée correspondante dans la table. Si la clé indiquée n'existe pas, la fonction lèvera une exception de type `KeyError`.
Pour vous aider, utilisez la méthode `pop()` sur les dictionnaires qui prend une clef en paramètre et qui supprime l'entrée correspondante dans le dictionnaire désigné.

Ex. 14.6. Nous vous demandons pour finir de rédiger un programme « chef d'orchestre » qui nous permet de créer, enregistrer et modifier une table dans une base de données à l'aide des fonctions définies précédemment. Il est demandé que le programme charge automatiquement la table sauvegardée au démarrage. Veillez à ce que la table soit sauvegardée aux bons moments de sorte que si nous quittons le programme, nous retrouvons bien toutes les données de la table.

```
(c)onsulter la table
(a)jouter une entrée
(m)ettre à jour une entrée
(s)upprimer une entrée
(q)uitter
SGBD>> s
Quel est le nom de l'étudiant à supprimer ?
Yami
L'étudiant recherché n'existe pas dans la table.
(c)onsulter la table
(a)jouter une entrée
(m)ettre à jour une entrée
(s)upprimer une entrée
(q)uitter
SGBD>> ...
```

Chapitre 15

Révision

Exercices préparatoires

Prép. 15.1. Revoir toute la matière :-)

Exercices en séance

Ex. 15.1. Écrivez une fonction `multiple9` qui prend un paramètre entier $n \in \{1, \ldots, 9\}$ et qui renvoie le produit de n par 9 sous forme d'une chaîne de caractères, sans utiliser l'opérateur de multiplication ni de boucles.

Ex. 15.2. Ecrivez une fonction `compress` correspondante à l'exercice 5.13

Ex. 15.3. Écrivez une fonction `deepprint(dico, depth)`, qui va afficher les éléments d'un dictionnaire en précisant pour chaque élément du dictionnaire la clé et sa valeur. Si la valeur est elle-même un dictionnaire, il faudra effectuer récursivement son affichage en indentant l'affichage ; le paramètre `depth` est en entier qui permet de préciser à quel niveau d'indentation on se situe (0 signifie pas d'indentation, 1 signifie qu'on a un niveau d'indentation, etc.). Voici un exemple d'affichage :

```
Nom: Pierre-Paul
Prénom: Jacques
Âge: 26
Cours:
  INFO-F-101:
    NoteFinale: 14
    InterroJanvier: 12
    Projets: 10
  INFO-F-102: 17
Matricule: 372910
```

Ex. 15.4. On stocke dans un fichier le nom et l'adresse de différentes personnes (une personne et son adresse par ligne). Par exemple :

```
David Roman#Rue des chateaux, 123#1100 Youplala#Belgique
Eric Degeers#Avenue de l'avenue, 13#11800 Mouahahah#France
John Meersman#Route de la vie, 55#1000 Forest#Belgique
...
```

Vous devez réaliser un programme qui permettra à l'utilisateur d'afficher ces informations selon différents tris :

– En fonction du nom, puis du code postal, puis du pays
– En fonction du pays, puis de la rue, puis du nom
– En fonction du code postal, puis du nom
Le programme devra continuellement demander à l'utilisateur comment il veut afficher les informations jusqu'à ce que ce dernier demande de stopper le programme.

Ex. 15.5. (Examen Septembre 2006, Info-f-206)
Un *nombre parfait* est un nombre entier naturel qui est égal à la somme de ses diviseurs, y compris 1, mais excepté lui-même. On vous demande d'écrire une fonction `nombre_parfait(nbr)` qui renvoie la valeur booléenne `true` si nbr est un nombre parfait et la valeur `false` sinon.
Exemples de nombres parfaits :

```
6   =  1 + 2 + 3\\
28  =  1 + 2 + 4 + 7 + 14\\
496 =  1 + 2 + 4 + 8 + 16 + 31 + 62 + 124 + 248
```

Ex. 15.6. (Examen bidon 2012, Info-f-206)
On considère des chaînes de caractères dans l'alphabet $\{A, C, G, T\}$, représentant des séquences ADN. Une telle chaîne est dite *complémentaire* d'une autre, si on obtient la deuxième en remplaçant chaque occurrence de la lettre A par T, T par A, G par C, et C par G dans la première. Par exemple, les séquences $ACCGAT$ et $TGGCTA$ sont complémentaires.

Ecrire une fonction qui reçoit en paramètre une liste de telles chaînes, et renvoie une nouvelle liste dans laquelle les paires complémentaires sont éliminées. Pour chaque paire complémentaire présente dans la liste de départ, seule la seconde chaîne de la paire sera conservée dans la liste résultat. Par exemple, la liste suivante :

```
['ATCC', 'ACGG', 'TGCC', 'ACC', 'TAGG']
```

contient deux paires complémentaires : `'ACGG'` et `'TGCC'` d'une part, et `'ATCC'` et `'TAGG'` d'autre part. On doit éliminer la première chaîne (dans l'ordre de la liste) de chaque paire. La liste renvoyée par la fonction sera donc :

```
['TGCC', 'ACC', 'TAGG']
```

On supposera que toutes les chaînes de la liste de départ sont distinctes.

Ex. 15.7. * (Examen Janvier 2013)
On vous demande d'implémenter la fonction `alphaCount()` qui prend en argument une chaîne de caractères `text`. Vous devez implémenter la fonction `alphaCount()` de deux façons différentes :

1. vous avez le droit d'utiliser tous les outils fournis par *Python*,
2. vous ne pouvez pas utiliser :
   – l'operateur `in`,
   – les fonctions `set()`, `list()`, `str()`,
   – les méthodes sur des strings, sur des sets ou sur des listes.

La fonction `alphaCount()` doit renvoyer un entier – le nombre de lettres différentes utilisées dans la chaîne `text`. La fonction `alphaCount()` ne doit pas compter les espaces, les nombres, ni les symboles de ponctuation. Les majuscules et les minuscules sont considérées comme la même lettre (e.g. les symboles "b" et "B" représentent la même lettre). Voir les exemples sur la Figure 15.1.

```
>>> alphaCount("bonjour")
>>> 6
>>> alphaCount("acacia")
>>> 3
>>> alphaCount("AaAaAa")
>>> 1
>>> alphaCount("Buvez de ce whisky que le patron juge fameux.")
>>> 26
>>> alphaCount("Monsieur Jack, vous dactylographiez bien mieux que votre ami Wolf!")
>>> 26
>>> alphaCount("Python 3 : alphaCount()")
>>> 10
```

FIGURE 15.1 – Exemple : les appels de la fonction `alphaCount()`.

Notez qu'il est possible d'implémenter la fonction `alphaCount()` avec une complexité en $O(len(text))$.
Vous pouvez supposer que tous les caractères sont en `ASCII` et que `text` ne contient pas de caractères accentués.

Ex. 15.8. On vous demande d'implémenter la fonction `tassement` qui prends en argument une liste `L` (qui contient des nombres naturels). La fonction doit modifier la liste `L` pour que tous les éléments 0 se retrouvent à la fin de la liste. On vous demande d'implémenter cette fonction de deux façons (voir les exemaples) :
– L'ordre de tous les éléments différents de 0 ne dois pas changer (`tassementOrd`),
– L'ordre de tous les éléments différents de 0 peut être modifié (`tassementNonOrd`)
Exemple :

```
>>> L = [1,4,2,0,4,0,3,2,0,0,1]
>>> tassementOrd(L)
>>> print(L)
>>> [1,4,2,4,3,2,1,0,0,0,0]
>>> L = [1,4,2,0,4,0,3,2,0,0,1]
>>> tassementNonOrd(L)
>>> print(L)
>>> [1,4,2,1,4,2,3,0,0,0,0]
```

Ex. 15.9. Écrivez une fonction `appartient` qui prend deux arguments, le premier est une chaîne de caractères, le deuxième est une liste de chaînes de caractères. La fonction vérifie l'appartenance de chaque élément de la liste dans le premier argument. Si le test est vérifié l'élément est ajouté à une première liste, sinon il est ajouté dans une autre liste. La fonction renvoie ensuite les deux listes crées. (Note : la fonction ne peut pas faire appel à la fonction `find`)

Ex. 15.10.* (Examen Janvier 2012)
La notation polonaise est aussi connue sous le nom de *notation préfixée*. L'idée de cette notation est simple : on écrit d'abord l'opérateur et ensuite les deux opérandes (par exemple « + 2 3 »), par opposition à la notation classique dite *infixe* où l'opérateur est placé entre les deux opérandes (par exemple « 2 + 3 »), voir Figure 15.2.
On vous demande d'écrire une fonction `evaluate` qui prend en paramètre une liste `expression` et qui renvoie un entier `result`, résultat de l'évaluation de l'expression représentée par `expression`. La liste `expression` est structurée de la façon suivante :

– chaque élément est un *string* ;
– chaque élément représente soit un entier, soit un opérateur ;
– les éléments se trouvent dans la liste suivant l'ordre de la notation polonaise (en d'autres termes, nous supposons que la liste est une expression valide).

Voici la liste des opérations possibles avec leurs représentations dans la liste :
– `"+"` : somme
– `"-"` : différence
– `"*"` : produit
– `"/"` : division entière
– `"%"` : reste de la division

```
Expression :
12 + 3 - 4 * 2
Notation polonaise correspondante :
- + 12 3 * 4 2
Représentation sous forme de liste :
["-", "+", "12", "3", "*", "4", "2"]
Résultat: 7
```

FIGURE 15.2 – Exemple : notation polonaise, représentation de l'expression sous forme d'une liste.

Vous pouvez supposer que la liste est correctement structurée, c'est-à-dire que l'ordre et le nombre d'éléments sont corrects. Votre fonction devra toutefois signaler par le biais d'une exception l'éventualité d'une division par zéro.
Essayez d'utiliser le moins de `if-else` possible.
Vous ne pouvez pas utiliser la fonction `eval()` de Python.
Vous pouvez définir d'autres fonctions et écrire du code en plus pour simplifier le code de la fonction `evaluate()`.
Voici encore quelques exemples de liste `expression` :

```
Expression:
1 + 1 + 1 + 1 + 1
Représentation sous forme d'une liste:
["+","+","+","+", "1", "1", "1", "1", "1"]
Résultat: 5

Expression:
((5 - 3) + (1 % 2)) * 8
Représentation sous forme d'une liste:
["*", "+", "-", "5", "3", "%", "1", "2", "8"]
Résultat: 24
```

FIGURE 15.3 – Exemples.

Ex. 15.11. * (Examen Janvier 2013)

Le problème de partition consiste à découper un ensemble d'entiers L en deux partitions L_1 et L_2 tel que la somme des éléments du premier ensemble soit égale à la somme des éléments du second.

Par exemple, supposons que L vaut `[10,4,9,5,3,7]`. Dans ce cas, une solution au problème de partition appliqué à L est L_1 qui vaut `[10,9]` et L_2 qui vaut `[4,5,3,7]`. En effet, la somme des éléments de L_1 est la même que la somme des éléments de L_2. Plus précisément la somme est égale à 19 dans les deux partitions.
On vous demande d'écrire une fonction partition qui résout le problème de partition. Plus précisément, votre fonction prendra en entrée une liste d'entiers L et renverra deux listes L_1 et L_2 à l'aide d'un tuple. L_1 et L_2 formeront une partition de L. De plus, la somme des éléments de L_1 devra être égale à la somme des éléments de L_2.
Notez que chaque ensemble d'éléments L n'a pas forcément de solution. Par exemple la liste `[4,5]` n'a pas de solution. Nous supposerons que la liste L donnée en entrée de votre fonction contient une solution.

Annexe A

Swampy Turtle

Exercices préparatoires

Matière à réviser :
– géométrie euclidienne de base ;
– angles supplémentaires, complémentaires ;
– périmètre d'un polygône, circonférence d'un cercle ;
– somme des angles dans un polygône régulier ;
– angles d'un polygône régulier.

Exercices en séance

Pour cette séance, vous devez d'abord télécharger *Swampy* pour Python 3 sur le site :

`http://swampy.googlecode.com/files/swampy.1.4.python3.zip`

Une fois le dossier décompressé, créez-y un fichier <*filename*>*.py*. Pour commencer, tapez le code suivant dans le fichier que vous venez de créer :

```
import turtle

turtle.reset()
```

Ce morceau de code ne fait que créer un monde et une tortue. Dans Turtle, chaque tortue peut avancer (`forward`), tourner à gauche (`left`) ou tourner à droite (`right`). De plus, sous chaque tortue se trouve un crayon permettant de tracer un trait si le crayon est baissé. La commande `up` permet de lever le crayon alors que la commande `down` permet de le baisser. Les fonctions `left` et `right` prennent un paramètre qui représente l'angle de rotation : `turtle.left(45)` fera tourner la tortue de 45 degrés vers la gauche.
Pour tracer un angle droit, il faut ajouter (après avoir créé la tortue `turtle`) le code suivant :

```
turtle.forward(100)
turtle.left(90)
turtle.forward(100)
```

Vérifiez que ces commandes dessinent bien un angle droit.

Ex. A.1. Modifier les commandes de sorte à dessiner un carré.

Ex. A.2. Écrivez une fonction `square` qui prend en paramètre une tortue `myTurtle` et qui lui fait dessiner un carré. Appelez ensuite cette fonction en lui passant la tortue `turtle` en paramètre.

Ex. A.3. Modifiez la fonction `square` en lui ajoutant un paramètre `length` qui représentera la longueur des côtés du carré. Testez votre fonction avec différentes longueurs.

Ex. A.4. Faites une copie de la fonction `square` et nommez-la `polygon`. Cette fonction prendra un paramètre supplémentaire `n` et devra dessiner un polygône régulier à `n` côtés.

Ex. A.5. Écrivez une fonction `circle` qui prend en paramètres une tortue `myTurtle` et un rayon `r`. Cette fonction dessinera un cercle de manière approximative à l'aide de la fonction `polygon` (astuce : faites en sorte que length $\times$ n = circonférence).

Ex. A.6. Écrivez une version plus spécifique de `circle`, appelée `arc`, qui prend un paramètre supplémentaire `angle` exprimé en degrés et déterminant la portion de cercle à tracer.

Ex. A.7.* (Examen de juin 2011) Il pourrait être utile, pour certaines applications, de pouvoir utiliser la tortue en précisant plutôt une liste de points dans le plan par laquelle elle doit passer. Notamment, une telle fonctionnalité permettrait de plus aisément utiliser Turtle pour dessiner des figures mathématiques décrites plus facilement sous forme d'équation polaire (reliant un rayon à un angle) que sous forme cartésienne (reliant une abcisse à une ordonnée), tels des cercles ou des spirales.
Nous vous demandons en premier lieu d'écrire une fonction `relier_points()` qui prendra en paramètres une tortue t ainsi qu'un nombre arbitraire de tuples de deux entiers (x_1, y_1), (x_2, y_2), …, (x_n, y_n) représentant des points dans le plan. L'idée de cette fonction sera de considérer que la position courante de la tortue est l'origine $(0, 0)$ et d'ensuite la faire passer dans l'ordre par tous les points (x_1, y_1), (x_2, y_2), etc. Pour ce faire, l'algorithme suivant peut être utilisé :

1. Soit $i = 0$
2. Considérer le point (x_i, y_i) dans la liste L avec $x_0 = y_0 = 0$
3. Calculer $(\Delta x, \Delta y) = (x_{i+1} - x_i, y_{i+1} - y_i)$
4. Calculer $d = \sqrt{(\Delta x)^2 + (\Delta y)^2}$
5. Si $\Delta x > 0$, calculer $\alpha = \text{arctg}(\frac{\Delta y}{\Delta x})$. Si $\Delta x < 0$, calculer $\alpha = \text{arctg}(\frac{\Delta y}{\Delta x}) + \pi$. Sinon (si $\Delta x = 0$), il y a trois autres cas à considérer : si $\Delta y = 0$, alors $\alpha = 0$; si $\Delta y > 0$, alors $\alpha = \frac{\pi}{2}$; sinon $\alpha = -\frac{\pi}{2}$.
6. Faire tourner la tortue de $\frac{360\alpha}{2\pi}$ degrés vers la gauche (`left`)
7. Avancer la tortue d'une distance d (`forward`)
8. Faire tourner la tortue de $\frac{360\alpha}{2\pi}$ degrés vers la droite (`rt`)
9. Incrémenter i et recommencer à l'étape 2 si $i \leqslant n$

Dans un second temps, nous vous demandons d'implémenter une fonction `spirale()` d'un seul paramètre k et qui renverra une liste de points permettant d'approximer une *spirale d'Archimède*. Celles-ci se caractérisent par une équation en coordonnées polaires de la forme $\rho(\theta) = k\theta$ où ρ est le rayon, θ est l'angle et k est un nombre réel strictement positif qui caractérise la spirale. Une propriété de ces objets mathématiques est d'avoir une distance partout égale à $2k\pi$ entre les bras de la spirale (formellement, $\rho(\theta + 2\pi) = \rho(\theta) + 2k\pi$). Pour obtenir une liste de points (partielle) à partir d'une équation quelconque en coordonnées polaires $\rho(\theta)$, l'algorithme suivant peut être utilisé :

1. Soit $\theta = 0$
2. Calculer $d = \rho(\theta)$

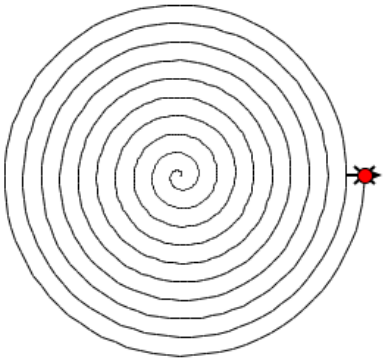
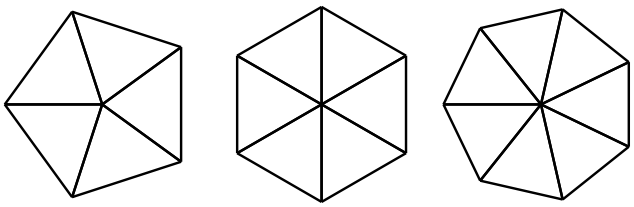


FIGURE A.1 – Spirale d'Archimède ($k = 2$) avec Turtle.

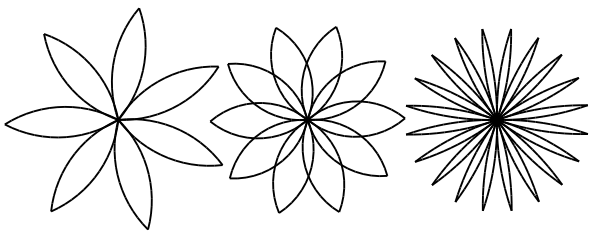
3. Calculer le point (x, y) correspondant par les relations $x = d\cos\theta$ et $y = d\sin\theta$
4. Augmenter θ de $\frac{\pi}{42}$ et recommencer à l'étape 2 jusqu'à ce que $\theta > 20\pi$.

Nous vous demandons enfin d'écrire une fonction `dessin_spirale()` ayant deux paramètres (une tortue t et le paramètre k de la spirale) et qui utilise les deux fonctions précédentes pour dessiner une spirale d'Archimède de paramètre k. La Figure A.1 montre à quoi ressemble le dessin d'une spirale d'Archimède de paramètre $k = 2$.
Pour vous assister, vous pouvez bien entendu faire appel aux objets adéquats du module `math` de Python, à savoir les fonctions `cos`, `sin` et `atan` (qui implémente arctg) ainsi que la constante `pi`. N'oubliez pas que les fonctions trigonométriques travaillent en radians et non en degrés.

Ex. A.8. ** Écrivez une fonction permettant de dessiner des figures telles :



Ex. A.9. ** Écrivez une fonction permettant de dessiner des figures telles :



Annexe B

Python Imaging Library

Exercices préparatoires

Matière à réviser :
– les commandes en terminal (`ls`, `cd`, `mkdir`, `cp`, `rm`, `rmdir` et autres).
– *Mode d'emploi introductif pour les salles du NO4 et NO3*, partie « console Linux ».

La bibliothèque PIL ne supporte pas encore Python 3, on travaillera donc avec la version 2 de Python : lancez `python2` plutôt que `python3` dans le terminal. Notez qu'en Python 2, la syntaxe de la fonction `print()` est légèrement différente : on écrit par exemple `print "Hello!"` au lieu de `print("Hello!")`. Une autre différence est le mode de fonctionnement de la division : en Python 3, l'opérateur `//` n'existe pas, le seul opérateur de division est `/`. Si les deux opérandes (numérateur et dénominateur) sont de type `int`, alors la division est entière (comme `//` en Python 3) ; si au moins un des deux opérandes est de type `float`, alors l'opérateur `/` effectue une division réelle (comme `/` en Python 3).

Prép. B.1. En utilisant l'interpréteur `python2`, faites l'exercice 1.3.

Prép. B.2. En utilisant l'interpréteur `python2`, faites l'exercice 1.4.

Prép. B.3. En utilisant l'interpréteur `python2`, faites l'exercice 1.6.

Prép. B.4. Pour cette séance, vous allez avoir besoin de plusieurs images en format `png`. A l'aide de votre éditeur d'images préféré, créez des images en format `png`. Par exemple, GIMP est un excellent logiciel libre d'édition d'images. Entre autres, préparez des images de tailles 500×500 pixels, 800×800 pixels, 800×600 pixels, 600×800 pixels et 100×100 pixels. N'oubliez pas d'apporter ces images au TP.

Prép. B.5. Que fait le code suivant ? Utilisez la fonction `help()` pour vous renseigner sur les fonctions utilisées.

```
import os

def test(x):
  return os.path.isfile(x) and x.split(".")[-1] == "png"

L = os.listdir(".")
L = list(filter(test, L))
print(L)
```

B.1 Notions de base

Nous allons vous demander de vous familiariser avec une bibliothèque qui vous permet de créer des programmes de traitement d'images. Pour importer la bibliothèque, ouvrir une image et l'afficher, il suffit d'effectuer la suite d'instructions suivantes :

```
>>> import Image
>>> picture = Image.open(nom_de_fichier)
>>> picture.show()
```

Ex. B.1. Créez un dossier de travail pour la séance, mettez-y des images ; ouvrez et affichez celles-ci via l'interpréteur. Utilisez des images en `.png`. Vous remarquerez que le principe est le même que pour l'ouverture d'un fichier via la fonction `open()`.

Ex. B.2. Les images possèdent des informations comme leur taille, leur mode de couleur, leur format, etc. Pour les récupérer, il suffit de faire :

```
>>> picture.size
(2446, 3720)
```

Consultez de la même manière les attributs `mode`, `format`, `format_description`, `filename` et déterminez à quoi correspondent ces informations.

Ex. B.3. Les images sont enregistrées avec la méthode `save`, comme montré dans l'exemple suivant :

```
>>> picture.save(nom_de_fichier, format_du_fichier)
```

On peut créer ce qu'on appelle des *thumbnails* (c'est-à-dire des miniatures) sur base d'un tuple déterminant la taille de la miniature.

```
>>> size = (94,94)
>>> picture.thumbnail(size)
```

Écrivez une fonction `traitement_par_lots(liste_noms, directory)` qui prend en paramètres une liste de noms de fichiers et un string contenant le chemin d'un dossier. La fonction devra ouvrir chaque fichier et en créer un *thumbnail* de taille 128×128 qui sera sauvegardé dans le dossier dont le chemin est fourni par `directory`.

B.2 Modifications et transformations géométriques

Vous pouvez également découper une image à partir d'un rectangle défini par les points supérieurs gauches et inférieurs droits. Ce rectangle est défini par un tuple contenant quatre valeurs `(x1,y1,x2,y2)` et doit être passé en paramètre à la méthode `crop(rectangle)`. Vous pouvez redimensionner l'image avec `resize(tuple)` et lui appliquer une rotation avec `rotate(degree)`. Vous pouvez appliquer des symétries orthogonales avec les appels à la méthode `transpose()` en lui passant, selon que vous désirez une symétrie d'axe vertical ou horizontal respectivement, `Image.FLIP_LEFT_RIGHT` ou `Image.FLIP_TOP_BOTTOM`. Voici un exemple :

```
>>> rectangle = (65,65,800,800)
>>> picture = picture.crop(rectangle)
>>> picture = picture.rotate(90)
>>> picture = picture.transpose(Image.FLIP_TOP_BOTTOM)
```

Ex. B.4. Écrivez une fonction `traitement_miroir(liste_noms)` qui prend en paramètre une liste de noms de fichiers. Vous avez fait une séance de shooting photo, malheureusement l'angle de vue était faite sur un miroir, l'appareil incliné de 30°. Nous vous demandons de traiter toutes ces photos dont la liste des fichiers est passée en paramètre. Ces photos doivent être recadrées avec un rectangle `(30,30,500,500)`, doivent être inversées par rapport à l'axe vertical et subir une rotation permettant de redresser l'image. Le résultat de ces manipulations pour chaque photo doivent remplacer le fichier d'origine.

B.3 Filtres

Vous pouvez également appliquer différents filtres d'image à l'aide du module `ImageFilter`, dont voici un exemple :

```
import ImageFilter

picture2 = picture.filter(ImageFilter.BLUR)
```

Il existe beaucoup d'autres modules de retouche. Si vous voulez aller plus loin, vous pouvez vous documenter sur la bibliothèque disponible à l'adresse suivante :

`http://www.pythonware.com/library/pil/`

Ex. B.5. Jouez maintenant avec la documentation et les différents filtres. Étudiez les modules `ImageFont` et `ImageDraw` ; essayez de les utiliser pour ajouter du texte dans une image.

Annexe C

Aide-mémoire Python

Cet aide-mémoire (éventuellement mis à jour) est disponible à chaque épreuve du cours. Vous pouvez également librement l'utiliser pendant les séances d'exercices. Le contenu de cette annexe correspond au formulaire disponible à l'examen de juin 2012, mis à jour pour être compatible avec Python 3.

C.1 Fonctions

- `int(x)` : convertit `x`, de type `float` ou `str`, en entier
- `float(x)` : convertit `x`, `int` ou `str`, en réel
- `str(x)` : convertit `x`, `int` ou `float`, en `str`
- `list(x)` : convertit `x` en `list`
- `tuple(x)` : convertit `x` en `tuple`
- `help(x)` : aide sur `x`
- `dir(x)` : liste des attributs de `x`
- `type(x)` : type de `x`
- **`print`**`(...)` : affiche
- `input(x)` : affiche le string `x` et lit le string qui est introduit au clavier
- `round(x)` : valeur arrondie du `float x`
- `len(s)` : longueur de la séquence `s`
- `range([start], stop, [step])` : renvoie une suite arithmétique d'entiers

C.2 Modules

- `math` : accès aux constantes et fonctions mathématiques (`pi`, `sin()`, `sqrt(x)`, `exp(x)`, `floor(x)` (valeur plancher), `ceil(x)` (valeur plafond), ...) : exemple : `math.ceil(x)`
- `copy` : `copy(s)`, `deepcopy(s)` : "shallow" et "deepcopy" de `s`
- `pickle` :
  - `dumps(v)` : transforme `v` en une représentation,
  - `loads(r)` : reconstitue l'objet
  - `dump(v,f)` : transforme et écrit dans le fichier `f`
  - `load(f)` : 'reconstitue à partir de la représentation lue de `f`
- `shelve` :
  - `db = open()` : créer un fichier comme objet de type `shelve`
  - `db.close()` : fermeture

C.3 Opérations et méthodes sur les séquences (str, list, tuples)

- `min(s)`, `max(s)` : élément minimum, maximum
- `sum(s)` : (ne fonctionne pas pour les string) : somme de tous les éléments (valeur numérique)
- `s.index(value, [start, [stop]])` : premier indice de value dans `s[start:stop]`
- `s.count(sub [,start [,end]])` : le nombre d'occurrences sans chevauchement de sub dans `s[start:end]`
- `map(f,s)` : créer une liste où chaque élément `i` de `s` est remplacé par `f(i)`
- `filter(f,s)` : créer une séquence du même type que `s` avec les éléments `i` de `s` tel que `f(i)` ou `i.f()` est vrai
- `it=iter(s)` : créé un itérateur

C.4 Méthodes sur les str

- `s.lower()` : met en minuscule
- `s.upper()` : met en majuscule
- `s.islower()`, `s.isdigit()`, `s.isalnum()`, `s.isalpha()`, `s.isupper()` : vrai si dans on a (respectivement) des minuscules, des chiffres, des car. alphanumériques, alphabétiques, majuscules
- `s.find(sub [,start [,end]])` : premier indice de `s` où le sous string `sub` est trouvé dans `s[start:end]`
- `s.replace( old, new[, co])` : renvoie une copie de `s` en remplaçant toutes les (ou les `co` premières) occurrences de `old` par `new`.
- `s.format(...)` : sert au formatage (en particulier) d'output
- `s.capitalize()` : met la première lettre en majuscule
- `s.strip()` : copie de `s` en retirant les « blancs » en début et fin
- `s.join(t)` : créer un `str` qui est le résultat de la concaténation des éléments de `t` chacun séparé par le str `s`
- `s.split([sep [,maxsplit])` : renvoie une liste d'éléments séparés dans `s` par le caractère `sep` (par défaut « blanc ») ; au max `maxsplit` séparations sont faites

C.5 Opérateurs et méthodes sur les listes

- `s.append(v)` : ajoute un élément valant `v` à la fin de la liste
- `s.extend(s2)` : rajoute à `s` tous les éléments de la liste `s2`
- `s.insert(i,v)` : insert l'objet `v` à l'indice `i`
- `s.pop()` : supprime le dernier élément de la liste et renvoie l''élément supprimé
- `s.remove(v)` : supprime la première valeur `v` dans `s`
- `s.reverse()` : renvoie la liste, le premier et dernier élément échange leurs places, le second et l'avant dernier, et ainsi de suite
- `s.sort(cmp=None, key=None, reverse=False)` : trie `s`
- `del s[i]`, `del s[i:j]` : supprime un ou des éléments de `s`
- `zip (a,b,c)` : construit une liste de des triples dont le i-ème élément reprend le i-ème élément de chaque séquence `a`,`b`,`c`

C.6 Méthodes sur les dict

- `d.clear()` : supprime tous les éléments de `d`

- `d.copy()` : “shallow” copie de `d`
- `d.fromkeys(s,v)` : créer un dict avec les clés de `s` et valeur `v`
- `d.get(k [,v])` : renvoie la valeur `d[k]` si elle existe, `V` sinon
- `d.items()` : liste des items `(k,v)` de `d`
- `d.keys()` : liste des clés
- `d.pop(k [,v])` : enlève `d[k]` et renvoie sa valeur ou `v`
- `d.popitem()` : supprimer un item (k,v) et renvoie l'item sous forme de tuple
- `d.setdefault(k [,v])` : `d[k]` si elle existe sinon `v` et rajoute `d[k]=v`
- `d.update(s)` : `s` est une liste de tuples que l'on rajoute à `d`
- `d.values()` : liste des valeurs de `d`

C.7 Méthodes sur les fichiers

- `f=open('fichier')` : ouvre `'fichier'` en lecture
- `f=open('fichier','w')` : ouvre `'fichier'` en écriture
- `f=open('fichier','a')` : ouvre `'fichier'` en écriture en rajoutant après les données déjà présentes
- `f.read()` : renvoie le contenu du fichier `f`
- `f.readline()` : lit une ligne
- `f.readlines()` : renvoie la liste des lignes de `f`
- `f.write(s)` : écrit la chaîne de caractères `s` dans le fichier `f`
- `f.close()` : ferme `f`

C.8 Exceptions

-
```
try:
    ...
    raise ...
    ...
  except:
    ...
  else:
    ...
  finally:
    ...
```